A High Performance Semantic Web Query Answering Engine

Description Logic Workshop ’05

Michael Wessel and Ralf Möller

Software Systems Group (STS)
Hamburg University of Technology (TUHH)
Hamburg, Germany

{mi.wessel | r.f.moeller}@tuhh.de
Overview of Talk

- Background & Requirements
- The nRQL Query Language
  - Introductory examples
  - Syntax (query atoms, variables, …)
  - Querying OWL Documents
  - Semantics
- The nRQL Query Processing Engine
  - Incremental Query Processing
  - Configurable Completeness
  - Simple Rules
  - Query Reasoning, Optimization
- Outlook & Conclusion
Racer(Pro) is an ABox DL reasoner for $\mathit{ALCQHI}_R^{+}(\mathcal{D}^-)$ aka $\mathit{SHIQ}(\mathcal{D}^-)$

Expressive ABox queries demanded by users

Question: how to design a query language which

- satisfies these user requests
- offers “full query access” to all Racer features (e.g., concrete domain)
- allows to query OWL documents (datatype & annotation properties, . . . )
- can be implemented efficiently
- has a simple orthogonal syntax and semantics

⇒ nRQL has been “tailored” for Racer
... what’s new since DL ’04

- more language features (e.g., a projection operator)
- serious implementation of query answering engine (concurrency control, thread pooling, ...)
- (preliminary) GUI support
  - tools for manipulating and inspecting the states of queries (RacerPorter)
  - life cycle management of queries
- (very) simple rules
- additional representation layers for Racer for storing data (rationale? see below)
- nRQL access to these additional layers
nRQL Language – Overview

- compositional syntax and semantics
- compound/complex queries build from query atoms, using boolean connectors
- allows for arbitrary shaped conjunctive queries
- Query atoms contain variables and individuals
- variables: \(?x, ?y\); individuals: betty

(retrieve (?x (told-value (age ?x)))
  (and (?x (and woman (an age)))
    (?x ?y has-child)
    (?y ?y (constraint
      (has-father age)
      (has-mother age)
      (> age-1 (+ age-2 8))))))
nRQL Language – Overview

- compositional syntax and semantics
- compound/complex queries build from query atoms, using boolean connectors
- allows for arbitrary shaped conjunctive queries
- Query atoms contain variables and individuals
- variables: $?x$, $??x$; individuals: betty

(((?x betty) ((told-value (age ?x)) 45))
  (((?x diana) ((told-value (age ?x)) 55))
   ...
  )}
nRQL Language – Example

True in all $\mathcal{I}$

Abstract Domain

Concrete Domain

"Betty"

45

20

55

"Betty"
nRQL Language – Example

Abstract Domain

Concrete Domain

True in all $\mathcal{I}$

betty

has_child

has_mother

charles

has_child

has_father

jane

age

name

betty-age

= 45

charles-age

= 55

jane-age

= 20

name

betty

= ’Betty”

betty-name

= 45

jane-age

= 20

charles-age

= 55

constraint

(retrieve (?x betty-age))

(retrieve (?x told-value (age ?x)))

(name age age age name)

(?x (and woman (an age)))

(?x ?y has-child)

(constraint (has-father age) (has-mother age) (> age-1 (+ age-2 8)))

(retrieve (?x (age ?x)) ...)

(retrieve (?x (told-value (age ?x))) ...)

DL ’05, 28th July 2005, Michael Wessel – p.6/43
Abstract Domain

Concrete Domain

Has child

Has father

Betty

Charles

Jane

$\text{has\_child}$

$\text{has\_father}$

$\text{name}$

$\text{age}$

$45$

$20$

$55$

$\text{Betty}$

$\text{charles}$

$\text{jane}$

$\text{betty\_age} = \text{Betty}$

$\text{charles\_age} = 45$

$\text{jane\_age} = 20$

$\text{betty\_age} = 55$

$\text{(?}\text{x (and woman (an age)))}$

$\text{(?}\text{x has\_child)}$

$\text{(?y (constraint (has\_father age) (has\_mother age) (> age-1 (+ age-2 8))))}$

$\text{true in all}$

$\text{(?}\text{x (and woman (an age)))}$

$\text{(?}\text{x ?y has\_child) (constraint (has\_father age) (has\_mother age) (> age-1 (+ age-2 8))))}$

$\text{retrieve (\text{age ?x}) ... (\text{told\_value (age ?x)}) ...}$

$\text{DL '05, 28th July 2005, Michael Wessel – p.6/43}$
Abstract Domain

Concrete Domain

nRQL Language – Example

?-x (and woman (an age))

(has-child age)

(has-father age)

(> age-1 (+ age-2 8))

(retrieve (?x (age ?x)) ...)

(retrieve (?x (told-value (age ?x))) ...)

(has-mother age)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)

(has-child)
nRQL Language – Example

Abstract Domain

Concrete Domain

(has-father age)
(has-mother age)
(> age-1
(+ age-2 8)))

\(\lambda(age_1, age_2). age_2 + 8 < age_1\)
Abstract Domain

Concrete Domain

\[ \lambda(\text{age}_1, \text{age}_2) \cdot 45 + 8 < 55 \Rightarrow \checkmark \]
nRQL Language – Example

Abstract Domain

Concrete Domain

(has_child ?x jane)

(has_mother jane betty)

(has_child betty charles)

(has_family jane charles)

(has_father charles jane)

(has_mother jane charles)

(has_mother betty jane)

(has_mother betty charles)

(name betty "Betty")

(age betty 45)

(age jane 20)

(age charles 55)

((?x betty) ((age ?x) betty-age))
nRQL Language – Example

Abstract Domain

Concrete Domain

(has_child ?x betty)
(has_father ?x jane)
(has_mother ?x charles)
(name ?x "Betty")
(age ?x 20)
(age jane 45)
(age charles 55)
((?x betty) ((told-value (age ?x)) 45))

(retrieve (?x (and woman (an age)))
  (?x ?y has-child)
  (?y ?y constraint
    (has-father age)
    (has-mother age)
    (> age-1 (+ age-2 8))))
  (retrieve (?x (age ?x)) ...)
  (retrieve (?x (told-value (age ?x))) ...)

True in all
nRQL Language – Syntax

- Queries have a head and a body:
  \[(\text{retrieve} \ <\text{head}> \ <\text{body}>)\]
- Syntax for \(<\text{head}>\)

\[
\begin{align*}
\text{head} & : = (\text{head\_entry}^*) \\
\text{head\_entry} & : = \text{object} \mid \text{head\_projection\_operator} \\
\text{object} & : = \text{variable} \mid \text{individual} \\
\text{variable} & : = \text{a symbol beginning with "?"} \\
\text{individual} & : = \text{a symbol} \\
\text{head\_projection\_operator} & : = (\text{cd\_attribute object}) \mid (\text{told\_value (cd\_attribute object)}) \mid (\text{told\_value (datatype\_property object)}) \mid (\text{annotations (annotation\_property object)})
\end{align*}
\]
nRQL Language – Syntax (2)

- Syntax for `<body>`
nRQL Language – Syntax (2)

- Syntax for `<body>`

\[
body \ := \ atom \mid (\{\text{and}\mid\text{union}\} \ body^*) \mid (\{\text{neg}\mid\text{inv}\} \ body) \mid (\text{project-to} \ (object^*) \ body)
\]
nRQL Language – Syntax (2)

- Syntax for `<body>`

\[
\text{body} \ := \ \text{atom} \ | \ \text{and} \ \text{body}^{*} \ | \ \text{union} \ \text{body}^{*} \ | \ \text{neg} \ \text{inv} \ \text{body} \ | \ \text{project-to} \ (\text{object}^{*}) \ \text{body}
\]
nRQL Language – Syntax (2)

- Syntax for `<body>`

\[
body \ := \ atom \ | \ (\{\text{and} \mid \text{union}\} \ body^*) \ | \ (\{\text{neg} \mid \text{inv}\} \ body) \ | \\
\text{(project-to} \ (\text{object}^*) \ body)\
\]
nRQL Language – Syntax (2)

- Syntax for `<body>`

\[
\text{body} := \text{atom} | (\{\text{and} | \text{union}\} \text{body}^*) | (\{\text{neg} | \text{inv}\} \text{body}) | \\
(\text{project-to} (\text{object}^*) \text{body})
\]
nRQL Language – Syntax (2)

- Syntax for `<body>`

\[
\text{body} \; := \; \text{atom} \mid (\{\text{and} | \text{union}\} \; \text{body}^*) \mid (\{\text{neg} | \text{inv}\} \; \text{body}) \mid \\
(\text{project-to} \; (\text{object}^*) \; \text{body})
\]

\[
\text{atom} \; := \; (\text{object} \; \text{concept_expr}) \mid (\text{object} \; \text{object} \; \text{role_expr}) \mid \\
(\text{object} \; \text{object} \; (\text{constraint} \; \text{chain} \; \text{chain} \; \text{constraint_expr})) \mid \\
(\text{same-as} \; \text{variable} \; \text{individual})
\]

\[
\text{chain} \; := \; (\text{role_expr}^* \; \text{cd_attribute})
\]

Example concept query atoms

- `(\text{x} \; \text{woman})`
- `(\text{betty} \; \text{woman})`
nRQL Language – Syntax (2)

• Syntax for `<body>`

\[
body \ := \ atom \mid (\{\text{and} \mid \text{union}\} \ body^* ) \mid (\{ \text{neg} \mid \text{inv}\} \ body ) \mid (\text{project-to} \ (\text{object}^* ) \ body )
\]

\[
atom \ := \ (\text{object concept_expr}) \mid (\text{object object role_expr}) \mid (\text{object object (constraint chain chain chain constraint_expr)}) \mid (\text{same-as variable individual})
\]

\[
chain \ := \ (\text{role_expr}^* \ cd_attribute)
\]

Example role query atoms

• `(？x ？y has-child)`

• `(betty ？child-of-betty has-child)`

• `(？x ？y (inv has-child))`

• `(？x ？y (not has-father))`
nRQL Language – Syntax (2)

- Syntax for `<body>`

\[
\text{body} \ := \ \text{atom} \ | \ (\{\text{and} \ | \ \text{union}\} \ \text{body}^*) \ | \ (\{\text{neg} \ | \ \text{inv}\} \ \text{body}) \ | \\
\ (\text{project-to} \ \text{(object}^*) \ \text{body})
\]

\[
\text{atom} \ := \ \text{object concept_expr} \ | \ \text{object object role_expr} \ | \\
\ (\text{object object} \ (\text{constraint chain chain chain constraint_expr})) \ | \\
\ (\text{same-as variable individual})
\]

\[
\text{chain} \ := \ \text{role_expr}^* \ \text{cd_attribute}
\]

Example constraint query atoms

- `(\?x \ ?y (constraint (has-mother age) (has-father age) <))`
- `(\?x \ ?y (constraint (has-brother age) (age) (= age-1 (+ 8 age-2))))`
nRQL Language – Syntax (2)

- Syntax for `<body>`

\[
body := \text{atom} | (\{\text{and} | \text{union}\} \text{body}^*) | (\{\text{neg} | \text{inv}\} \text{body}) | (\text{project-to} (\text{object}^*) \text{body})
\]

\[
\text{atom} := (\text{object concept_expr}) | (\text{object object role_expr}) | (\text{object object (constraint chain chain constraint_expr)}) | (\text{same-as variable individual})
\]

\[
\text{chain} := (\text{role_expr}^* \text{cd_attribute})
\]

Example same-as query atoms

- (same-as ?x betty)
nRQL Language – Syntax (2)

• Syntax for `<body>`

\[
\begin{align*}
\text{body } & := \text{ atom } \mid (\{\text{and} \mid \text{union}\} \text{ body}^*) \mid (\{\text{neg} \mid \text{inv}\} \text{ body}) \mid \\
& \quad (\text{project-to} (\text{object}^*) \text{ body}) \\
\text{atom } & := (\text{object concept_expr}) \mid (\text{object object role_expr}) \mid \\
& \quad (\text{object object} (\text{constraint} \text{ chain} \text{ chain constraint_expr})) \mid \\
& \quad (\text{same-as variable individual}) \\
\text{chain } & := (\text{role_expr}^* \text{ cd_attribute}) \\
\text{concept_expr} & := \text{ a Racer concept, with some extensions for OWL} \\
\text{role_expr} & := \text{ a Racer role} \mid (\text{inv role_expr}) \mid (\text{not role_expr}) \\
\text{constraint_expr} & := \text{ a Racer concrete predicate} \\
\text{cd_attribute} & := \text{ a Racer concrete domain attribute} \\
\text{datatype_property} & := \text{ a Racer role used as OWL datatype property} \\
\text{annotation_property} & := \text{ a Racer role used as OWL annotation property}
\end{align*}
\]
nRQL Variables

- Variables can only be bound to ABox individuals, not to concrete domain objects or even concrete domain values

- nRQL offers two kinds of variables: $x, $x
  - $x prohibits binding to individuals which are already bound by other variables, e.g. $y (mapping must be injective)

- “UNA” for variables

  ? (retrieve ($x $y) (and ($x man) ($y man)))
  > NIL

  ? (retrieve ($?x $?y) (and ($?x man) ($?y man)))
  > (($?X CHARLES) ($?Y CHARLES))
Complex nRQL Queries

- Compound nRQL queries are defined inductively
  - Every query atom $a_i$ is a body.
  - If $a_1 \ldots a_n$ are query bodies, then the following expressions are also bodies
    - $(\text{neg } a_i)$
    - $(\text{inv } a_i)$
    - $(\text{and } a_1 \ldots a_n)$
    - $(\text{union } a_1 \ldots a_n)$
    - $(\text{project-to } (\text{objects-in-ai}) \ a_i)$

- Each variable creates a new axis in an $n$-dimensional tuple space

- A projection (specified by `<head>`) is made before that set is returned.
Illustration of Semantics

(retrieve (?x)
  (and (?x woman)
   (?y man)
   (?x ?y has-child)))
Illustration of Semantics

(retrieve (?x)
  (and
    (?x woman)
    (?y man)
    (?x ?y has-child)))
Illustration of Semantics

(retrieve (?x)
  (and (and (?x woman) (?y top))
   (?y man)
   (?x ?y has-child)))
Illustration of Semantics

(retrieve (?x)
  (and (and (?x woman) (?y top))
    (?y man)
    (?x ?y has-child)))
Illustration of Semantics

(retrieve (?x)
  (and (and (?x woman) (?y top))
    (and (?x top) (?y man))
    (?x ?y has-child)))
Illustration of Semantics

(retrieve (?x)
  (and
    (and (?x woman) (?y top))
    (and (?x top) (?y man))
    (?x ?y has-child)))
Illustration of Semantics

(retrieve (?x)
  (and (and (?x woman) (?y top))
       (and (?x top) (?y man))
       (?x ?y has-child)))
Illustration of Semantics

(retrieve (?x)
 (and (and (?x woman) (?y top))
 (and (?x top) (?y man))
 (?x ?y has-child)))
Illustration of Semantics

(retrieve (?x)
  (and (and (?x woman) (?y top))
   (and (?x top) (?y man))
   (?x ?y has-child)))
Negation in nRQL

- nRQL offers NAF with \texttt{neg}
- Semantics: simple set complement
  - well-defined for concept and role query atoms
  - well-defined for compound queries (DeMorgan etc.)
  - some “tricks” are needed for \texttt{same-as} and \texttt{constraint} query atoms
- Classical DL-like negation
  - obviously, in concept query atoms
  - but also in role query atoms
    \((?x \ ?y \ (\texttt{not} \ \texttt{has-father}))\)
Illustration of \texttt{neg}

\begin{code}
(retrieve (?x)
  (and (?x woman)
    (and (?y man)
      (?x ?y has-child))))
\end{code}
Illustration of \texttt{neg}

\texttt{(retrieve (?x)}

\texttt{(neg (and (?x woman)}

\texttt{?y (\texttt{x} ?y has-child)))}
Illustration of \textbf{neg}

\begin{align*}
\text{(retrieve} (\textit{?x}) \\
\text{(union} (\text{neg} \ (\textit{?x} \ \text{woman})) \\
\text{(neg} \ (\textit{?y} \ \text{man})) \\
\text{(neg} \ (\textit{?x} \  \textit{?y} \ \text{has-child}))))
\end{align*}
Illustration of \texttt{neg}

(retrieve (?x)

(union (neg (?x woman))

\textbf{(neg (?y man))}

(neg (?x ?y has-child))))

\textbf{Illustration of neg}

(retrieve (?x)

(union (neg (?x woman))

\textbf{(neg (?y man))}

(neg (?x ?y has-child))))
Illustration of \textbf{neg}

\[
\begin{align*}
\text{retrieve (}\ ?x) \\
\text{(union (neg (}\ ?x \ \text{woman})) \\
\quad (\text{and (}\ ?x \ \text{top}) \ (\text{neg (}\ ?y \ \text{man})))) \\
\quad (\text{neg (}\ ?x \ ?y \ \text{has-child})))
\end{align*}
\]
Illustration of \textit{neg}

\begin{verbatim}
(retrieve (?x)
  (union (?x woman)
    (?y (and (?x top) (neg (?y man)))
     (neg (?x ?y has-child))))
\end{verbatim}
Illustration of \texttt{neg}

\begin{equation*}
\text{(retrieve } (?x) \text{)}
\end{equation*}

\begin{equation*}
\text{(union}
\begin{align*}
&\text{(and } (\text{neg } (?x \text{ woman}) \text{ } (?y \text{ top}))\text{)} \\
&\text{(and } (?x \text{ top}) \text{ (neg } (?y \text{ man}))\text{)} \\
&\text{(neg } (?x \text{ } ?y \text{ has-child}))\text{)}
\end{align*}
\end{equation*}
Illustration of $\text{neg}$

$$(\text{retrieve } (?x))$$

$$(\text{union } (\text{and } (\text{neg } (?x \text{ woman}) (?y \text{ top})))$$

$$(\text{and } (?x \text{ top}) (\text{neg } (?y \text{ man})))$$

$$(\text{neg } (?x \ ?y \text{ has-child})))$$
Illustration of \textit{neg}

(retrieve (?x)
  (and (?x woman)
    (?y man)
    (?x ?y has-child)))
Illustration of $\text{neg}$

$\text{(retrieve } (?x) \text{)}$

$\text{neg} \ (\text{and } (?x \text{ woman)}$

$\text{(?y \text{ man)}$

$\text{(?x ?y has-child)}}))$
Illustration of \textbf{neg}

\begin{enumerate}
\item \textit{(retrieve}\ $\mathbf{(?x)}$
\item \textit{(neg (and $\mathbf{(?x woman)}$}
\item \textit{($\mathbf{(?y man)}$}
\item \textit{($\mathbf{(?x ?y has-child)}$))}
\end{enumerate}
Illustration of $\text{neg}$

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

$(\text{and} \ (?x \ \text{woman})$ $\ \ ?y \ \ ?y \ \text{man})$

$(?x \ ?y \ \text{has-child}))$
Illustration of \textit{neg}

\[
\text{NEG} \ (\text{retrieve} \ (?x) \\
\quad \text{(and} \ (?x \ \text{woman}) \\
\quad \quad (?y \ \text{man}) \\
\quad \quad (?x \ ?y \ \text{has-child})))))
\]
Illustration of \textit{neg} \\

(retrieve (?x) \\
(neg (project-to (?x) \\
?y (and (?x woman) (?y man) \\
(?x ?y has-child)))))
Queries with Individuals

? (retrieve (betty)
    (betty woman))

> (((($?BETTY BETTY))))

- Explanation: query is rewritten into

  (AND (SAME-AS $?BETTY BETTY)
       ($?BETTY WOMAN))

- $?BETTY is a variable that does not obey the unique name assumption for variables
- (SAME-AS $?BETTY BETTY) enforces binding of $?BETTY to BETTY
Semantic Consequences

- “NAF” for atoms with individuals can be tricky

\[
\text{(retrieve (betty)}
\text{ (neg (betty woman))})
\]
\[
= 
\text{(retrieve ($?betty)}
\text{ (neg (and ($?betty woman)
\text{ (same-as $?betty betty))}}
\]
\[
= 
\text{(retrieve ($?betty)}
\text{ (UNION (neg ($?betty woman))}
\text{ (neg (same-as $?betty betty)))}
\]

- one **must** define the semantics in such a way if the orthogonality of the language shall be preserved!
The Projection Operator

• We can retrieve all woman having children with

\[
\text{retrieve (} ?x \text{)} \\
\text{(and (} ?x \text{ woman) (} ?x \ ?y \text{ has-child))}
\]

• How can we retrieve woman which have no (known) children?

\[
\text{retrieve (} ?x \text{)} \\
\text{(and (} ?x \text{ woman) (all has-child bottom))}
\]

\[
\text{retrieve (} ?x \text{)} \\
\text{(and (} ?x \text{ woman) (neg (} ?x \ ?y \text{ has-child))}}
\]

\[
\text{retrieve (} ?x \text{)} \\
\text{(neg (and (} ?x \text{ woman) (} ?x \ ?y \text{ has-child))})}
\]
The Projection Operator (2)

Q1: (retrieve (?x)
    (and (?x woman)
    (neg (?x ?y has-child))))
The Projection Operator (2)

Q1: (retrieve (?x)
(and (?x woman)
(neg (?x ?y has-child)))))
The Projection Operator (2)

Q1: (retrieve (?x)
  (and (?x woman)
   (neg (?x ?y has-child)))))
The Projection Operator (2)

\[ Q1: \text{(retrieve} \ (\text{\(?x\))} \]
\[
\text{(and \ (?x \ woman)}
\[
\text{(neg \ (?x \ ?y \ has-child)))} \]
Q2: (retrieve (?x)
   (neg (and (?x woman)
     (?x ?y has-child))))
The Projection Operator (2)

?- Q2: (retrieve (?x)
    (neg (and (?x woman)
            (?x ?y has-child))))

Diagram showing a grid with arrows indicating the projection of elements.
The Projection Operator (2)

Q3: (retrieve (?x)
    (and (?x woman)
        (neg (project-to (?x)
            (?x ?y has-child)))))
The Projection Operator (2)

Q3: (retrieve (?x)

(and (?x woman)

(neg (project-to (?x)

(?x ?y has-child)))))

?x

?y
The Projection Operator (2)

Q3: (retrieve (?x)
    (and (?x woman)
        (neg (project-to (?x) (?x ?y has-child))))))
The Projection Operator (2)

Q3: (retrieve (?x)
   (and (?x woman)
    (neg (project-to (?x)
           (?x ?y has-child)))))
The Projection Operator (2)

Q3: (retrieve (?x)
    (and (?x woman)
        (neg (project-to (?x)
            (?x ?y has-child)))))

\[ ?x \]

\[ ?y \]
...Some Syntactic Sugar

- Due to the new projection operator, some “special syntax” from the older nRQL (DL ’04) can now be expressed

  - $(?x \ (\text{has-known-successor}\ \text{has-child}))$
    $= (\text{project-to}\ (?x)\ (?x\ ?y\ \text{has-child}))$

  - $(?x\ \text{NIL} \ \text{has-child})$ (borrowed from LOOM)
    $= (\text{neg} (\ ?x\ (\text{has-known-successor}\ \text{has-child})))$
    $= (\text{neg} (\text{project-to}\ (?x)\ (?x\ ?y\ \text{has-child})))$

- now expressible in terms of $\text{project-to}$
Querying OWL KBs

- **OWL datatype properties:**

```xml
<owl:Class rdf:ID="Person">
  <rdfs:label>person</rdfs:label>
</owl:Class>

<owl:DatatypeProperty rdf:ID="age">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

<Person rdf:about="http://www.test.com/michael">
  <age>34</age>
</Person>
```
nRQL & Datatype Properties

- Idea: handle OWL DTP like concrete domain attributes

\[ \text{? (retrieve} \]
\[ \quad (?x \]
\[ \quad \quad (\text{datatype-fillers} \]
\[ \quad \quad \quad (|\text{http://www.test.com/test.owl#age}| \ ?x))) \]
\[ \quad (?x \ (\text{some} \ |\text{http://www.test.com/test.owl#age}| \]
\[ \quad \quad \quad \quad (\text{and} \ (\text{min} \ 30) \ (\text{max} \ 35)))))) \]
\[ \quad \quad > \ (((?X \ |\text{http://www.test.com/michael}|) \]
\[ \quad \quad \quad (\text{:TOLD-VALUE} \]
\[ \quad \quad \quad \quad (|\text{http://www.test.com/test.owl#age}| \ ?X)) \ (34)))) \]

- Extended Racer concept syntax (expressions like (and (min 30) (max 35)) only recognized by nRQL)
nRQL & Annotation Properties

```
<owl:AnnotationProperty rdf:ID="my-comment">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#person"/>
</owl:AnnotationProperty>

<person rdf:ID="i">
  <my-comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    My comment
  </my-comment>
</person>
```

- A special head projection operator annotations (told-value) is provided by nRQL
- Similar to querying for datatype properties
Expressivity Problems

- Access to “data values” in OWL docs (fillers of datatype/annotation properties) is restricted
- from the DL perspective, only the (extended) Racer concept expression language can be used
- How to retrieve all individuals which have (CD attribute or DTP) fillers containing substring $x$?
- Solution: maintain a data substrate in parallel to an ABox
- the data substrate is used to automatically “mirror” the ABox
- offer query access to this substrate by means of a hybrid query language - nRQL
Hybrid Queries

(retrieve (?x ?*name ?*age)
  (and (?x (and |http://...#person|
    (an |http://...#age|)))
  (?*x ?*name |http://...#name|)
  (?*name ( (:predicate (search "wessel"))
    ((:predicate (search "michael"))
      (:predicate (search "achim")))))
  (?*x ?*age |http://...#age|)
  (?*age ((:predicate (< 40)))))

- New sort of variables: *?x (*$?x), ranging over data nodes
- Data nodes can also be data values in OWL documents
- Data nodes/edges have descriptive labels: kind, role, property, ...
- Notion of entailment for labels of nodes/edges
- Data query atoms are in pos. CNF & contain literals and predicates.
Formal Semantics - Auxiliaries

- The projection
  \( T' =_{def} \{ \langle t_{i_1}, \ldots, t_{i_m} \rangle \mid \langle t_1, \ldots, t_n \rangle \in T \} = \pi_{<i_1,\ldots,i_m>}(T) \) of \( T \) to the components mentioned in the index vector \( \langle i_1, \ldots, i_m \rangle \).
  
  Example:
  \[ \pi_{<1,3>}(\{ \langle 1, 2, 3 \rangle, \langle 2, 3, 4 \rangle \}) = \{ \langle 1, 3 \rangle, \langle 2, 4 \rangle \}. \]

- If \( \vec{b} \) is a bit vector which contains exactly \( m \) ones, and \( B \) is a set, \( T \) is a set of \( m \)-ary tuples, then
  
  the \textit{n-dimensional extension} \( T' \) of \( T \) w.r.t. \( B \) and \( \vec{b} \) is defined as
  \[ T' =_{def} \{ \langle i_1, \ldots, i_n \rangle \mid \langle j_1, \ldots, j_m \rangle \in T, 1 \leq l \leq m, 1 \leq k \leq n \}
  \]
  with \( i_k = j_l \) if \( b_k = 1 \), and \( b_k \) is the \( l \)th one (1) in \( \vec{b} \),
  otherwise, \( i_k \in B \).

  and denoted by \( \chi_{B,<b_1,\ldots,b_n>}(T) \).

  Example:
  \[ \chi_{\{a,b\},<0,1,0,1>}(\{ \langle x, y \rangle \}) = \{ \langle a, x, a, y \rangle, \langle a, x, b, y \rangle, \langle b, x, a, y \rangle, \langle b, x, b, y \rangle \}. \]
Formal Semantics - Atoms

\[ (q'_{x_i} \text{ concept} \_expr)\mathcal{E} \defeq \chi_{\text{Inds}_A, \bar{I}_n, \{i\}}(\text{concept} \_\text{instances}(A, \text{concept} \_expr)) \]

\[ (q'_{x_i} q'_{x_j} \text{ rolen} \_expr)\mathcal{E} \defeq \chi_{\text{Inds}_A, \bar{I}_n, \{i, j\}}(\text{role} \_\text{pairs}(A, \text{role} \_expr)), \text{ if } i \neq j \]

\[ (q'_{x_i} q'_{x_i} \text{ role} \_expr)\mathcal{E} \defeq \chi_{\text{Inds}_A, \bar{I}_n, \{i\}}(\text{role} \_\text{pairs}(A, \text{role} \_expr) \cap \mathcal{I}D_{2, \text{Inds}_A}) \]

\[ (\text{same-as } q'_{x_i} \text{ ind})\mathcal{E} \defeq \chi_{\text{Inds}_A, \bar{I}_n, \{i\}}(\{\text{ind}\}) \]

\[ (q'_{x_i} q'_{x_j} (\text{constraint attrib}_1 \text{ attrib}_2 P))\mathcal{E} \defeq \chi_{\text{Inds}_A, \bar{I}_n, \{i, j\}}(\text{predicate} \_\text{pairs}(A, \text{attrib}_1, \text{attrib}_2, P)), \text{ if } i \neq j \]

\[ (q'_{x_i} q'_{x_i} (\text{constraint attrib}_1 \text{ attrib}_2 P))\mathcal{E} \defeq \chi_{\text{Inds}_A, \bar{I}_n, \{i\}}(\text{predicate} \_\text{pairs}(A, \text{attrib}_1, \text{attrib}_2, P) \cap \mathcal{I}D_{2, \text{Inds}_A}) \]
Formal Semantics - Bridge2DL

- Semantics of DL standard ABox retrieval functions ("Bridge to Racer’s basic ABox retrieval functions")

\[
\text{concept\_instances}(A, \text{concept\_expr}) = \{ i \mid i \in \text{Inds}_A, (A, T_A) \models \text{concept\_expr}(i) \}
\]

\[
\text{role\_pairs}(A, \text{role\_expr}) = \{ \langle i, j \rangle \mid i, j \in \text{Inds}_A, (A, T_A) \models \text{role\_expr}(i, j) \}
\]

\[
\text{predicate\_pairs}(A, \text{attrib}_1, \text{attrib}_2, P) = \{ \langle i, j \rangle \mid i, j \in \text{Inds}_A, (A, T_A) \models \exists x, y : \text{attrib}_1(i, x) \land \text{attrib}_2(j, y) \land P(x, y) \}
\]
Formal Semantics - Bodies

\[(\text{and} \ q_1 \ldots q_i)^\mathcal{E} = \{ \bigcap_{1 \leq j \leq i} q_j^\mathcal{E} \} \]
\[(\text{union} \ q_1 \ldots q_i)^\mathcal{E} = \{ \bigcup_{1 \leq j \leq i} q_j^\mathcal{E} \} \]
\[(\text{neg} \ q)^\mathcal{E} = \{ (\text{Inds}_A)^n \setminus q^\mathcal{E} \} \]
\[(\text{inv} \ q)^\mathcal{E} = \{ \text{inv}(q^\mathcal{E}) \text{, where inv reverses all tuples} \} \]
\[(\text{project-to} \ (x_{i_1},q \ldots x_{i_k},q))^\mathcal{E} = \{ \pi_{i_1,\ldots,i_k}(q^\mathcal{E}) \} \]

- Claim: the given semantics is easy to catch
  - only basic set-theory required
  - easy to visualize
  \[ \Rightarrow \] good for users
Features of the nRQL Engine

- Integral part of RacerPro

  ⇒ no communication overhead with Racer (an “external” query answering engine would have to communicate a lot with Racer, performance comparable to nRQL’s performance would be unachievable)

- “Multi-query” answering (multi-threaded)

- Different query processing modes

- Degree of completeness configurable

- Non-recursive defined queries (macro queries)

- Simple rule engine

- Semantic & cost-based Query Optimizer
Query Processing Modes

- **Set-at-a-time mode**
  - synchronous mode of interaction, call to `retrieve` blocks until answer is computed, returned as a bunch

- **Tuple-at-a-time mode**
  - asynchronous mode of interaction, call to `retrieve` returns immediately with query identifier
  - query thread works in the background
  - `get-next-tuple <id>` returns next tuple of query `<id>`
  - **Lazy**: compute next tuple if requested
  - **Eager**: precompute next tuple(s)
Degree of Completeness

- Mode 0: syntactic told information is used for query answering
- Mode 1: Mode 0 + exploited TBox information
- Mode 3: complete Racer ABox retrieval (expensive!)
- \( 3 \times \# \{set\_at\_a\_time, tuple\_at\_a\_time \} = 6 \)
- Variations: realize ABox / classify TBox (or not)
- Even more modes: “two-phase query 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)
Two-Phase Query Processing

**TBox:**

\[
\begin{align*}
\text{person} & \subseteq \top \\
\text{man} & \subseteq \text{person} \\
\text{woman} & \subseteq \text{person} \\
\text{spouse} & = \text{woman} \sqcap \\
& \quad (\exists \text{married\_to}.\text{man})
\end{align*}
\]

**ABox:**

\[
\begin{align*}
\text{spouse}(\text{doris}) \\
\text{spouse}(\text{betty}) \\
\text{man}(\text{adam}) \\
\text{woman}(\text{eve}) \\
\text{married\_to}(\text{eve}, \text{adam})
\end{align*}
\]

- (retrieve (?x) (?x spouse))
- (get-next-tuple :query-1)
- (get-next-tuple :query-1)

\[
\Rightarrow (\text{QUERY-1} :\text{RUNNING})
\]

\[
\Rightarrow ((?X \text{ DORIS}))
\]

DL ’05, 28th July 2005, Michael Wessel – p.31/43
Two-Phase Query Processing (2)

**TBox:**

- $person \subseteq \top$
- $man \subseteq person$
- $woman \subseteq person$
- $spouse \models woman \sqcap (\exists married\_to. man)$

**ABox:**

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

- $(\text{retrieve } (?x) \; (?x \; \text{spouse}))$

$\Rightarrow$ $(\text{:QUERY-1 :RUNNING})$

- $(\text{get-next-tuple :query-1})$

$\Rightarrow$ $( (?x \; BETTY))$
Two-Phase Query Processing (3)

**TBox:**

\[
\begin{align*}
\text{person} & \subseteq \top \\
\text{man} & \subseteq \text{person} \\
\text{woman} & \subseteq \text{person} \\
\text{spouse} & \models \text{woman} \sqcap \\
(\exists \text{married_to.man}) & \models \text{married_to}(\text{eve}, \text{adam})
\end{align*}
\]

**ABox:**

\[
\begin{align*}
\text{spouse(\text{doris})} \\
\text{spouse(\text{betty})} \\
\text{man(\text{adam})} \\
\text{woman(\text{eve})} \\
\text{married_to(\text{eve}, \text{adam})}
\end{align*}
\]

- \((\text{retrieve} (\text{x}) (\text{x spouse}))\)

\[\Rightarrow \text{(:QUERY-1 :RUNNING)}\]

- \((\text{get-next-tuple :query-1})\)

\[\Rightarrow \text{:WARNING-EXPENSIVE-PHASE-TWO-STARTS}\]
Two-Phase Query Processing (4)

TBox:

- person ⊆ ⊤
- man ⊆ person
- woman ⊆ person
- spouse ≡ woman ∩

(∃married_to.man)

• (retrieve (?x) (?x spouse))
⇒ (:QUERY-1 :RUNNING)

• (get-next-tuple :query-1)
⇒ ((?X EVE))

ABox:

- spouse(doris)
- spouse(betty)
- man(adam)
- woman(eve)
- married_to(eve, adam)
Two-Phase Query Processing (5)

TBox:

\[
\begin{align*}
\text{person} & \subseteq \top \\
\text{man} & \subseteq \text{person} \\
\text{woman} & \subseteq \text{person} \\
\text{spouse} & \equiv \text{woman} \sqcap \\
(\exists \text{married_to} . \text{man}) & \\
\end{align*}
\]

ABox:

\[
\begin{align*}
\text{spouse(doris)} \\
\text{spouse(betty)} \\
\text{man(adam)} \\
\text{woman(eve)} \\
\text{married_to(eve, adam)} \\
\end{align*}
\]

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

⇒ (:QUERY-1 :RUNNING)

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

⇒ :EXHAUSTED
Two-Phase Query Processing (6)

TBox:

\[
\begin{align*}
\text{person} & \subseteq \top \\
\text{man} & \subseteq \text{person} \\
\text{woman} & \subseteq \text{person} \\
\text{spouse} & \models \text{woman} \sqcap \\
\exists \text{married}_\text{to}.\text{man} & \\
\end{align*}
\]

ABox:

\[
\begin{align*}
\text{spouse}(\text{doris}) \\
\text{spouse}(\text{betty}) \\
\text{man}(\text{adam}) \\
\text{woman}(\text{eve}) \\
\text{married}_\text{to}(\text{eve}, \text{adam})
\end{align*}
\]

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

\[\Rightarrow (~\text{QUERY-1} ~\text{RUNNING})\]

- (get-answer :query-1)

\[\Rightarrow ((?X \text{ DORIS}) (\text{?X BETTY}) (\text{?X EVE}))\]
Optimization & Caching

- Caching of Racer results (cache consistency, ...)
- Lots of index structures (must be maintained, ...)
- Cost-based optimizer (reordering of conjuncts and marking variables as non-generators, e.g. ?y in \( \text{retrieve} (?x) (?x \ ?y \text{ has-child}) \))
- Reasoning with Queries (optional, incomplete)
  - Query consistency check
  - Query entailment check (subsumption)
  \( \Rightarrow \) maintenance of a “Query repository” DAG (similar to a taxonomy)
  - Query “realization” (adds implied conjuncts to enhance informedness of backtracking search)
Defined Queries

- nRQL offers a simple macro-mechanism

(defquery mother-of-child
  (?x ?y)
  (and (?x woman)
       (?x ?y has-child)))

(defquery mother-of-son
  (?x ?child)
  (and (?x ?child mother-of-child)
       (?child man)))

- no cyclic definitions allowed
Simple Rules

- nRQL offers a simple rule mechanism

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

- Rule antecedence is a query body; consequence is a list of generalized ABox assertions
- rules must be fired manually
Complex TBox Queries

- “What are the child (parent, descendant, ancestor) concepts of the concept woman”?
- Idea: view the taxonomy of a TBox as a relational structure (a DAG), stored as a “data substrate”
- use nRQL to query this structure with `tbox-retrieve`:

  \[
  \text{? (tbox-retrieve (?x ?y)}
  \begin{align*}
  & \quad \text{(and (?x woman)} \\
  & \quad \quad \text{(?x ?y has-child)))}
  \end{align*}
  \]

  \[
  > \quad (((?X WOMAN) (?Y SISTER)) \\
  & \quad (((?X WOMAN) (?Y AUNT)) \\
  & \quad (((?X WOMAN) (?Y *BOTTOM*)) \\
  & \quad (((?X WOMAN) (?Y MOTHER)) \\
  & \quad (((?X WOMAN) (?Y GRANDMOTHER))))
  \]

DL '05, 28th July 2005, Michael Wessel – p.40/43
Query Processing

• In principle, nRQL uses top-down query evaluation strategy:
  • each query evaluation plan determines the role of an atom as a generator or tester
  • optimizer: try to minimize required generators
  • in the presence of project-to, this becomes more involved
    • sub-queries must be evaluated first, e.g., in case of (neg (project-to ...))
    ⇒ bottom-up-top-down mixture of query evaluation
  • continuation-style implementation, can be compiled (see implementations of Prolog in Lisp)
Related Activities & Conclusion

- Benchmarking nRQL: wait for Ralf’s talk
- nRQL as a basis for a subset of OWL-QL: Atila’s & Jan’s Poster
- “nRQL tab” for Protégé (Kruthi & Volker)
- RacerPorter supports life-cycle management and inspection of nRQL queries and rules
- only nRQL implementation: 29.553 LOC

Future plans
- rolling-up (support OWL-QLs “do-not-bind variables” for acyclic conjunctive queries)
- better index structures for data substrate layer
- database access
Thanks for your attention!