Expressivity and Complexity of MongoDB queries

Elena Botoeva

Faculty of Computer Science, Free University of Bozen-Bolzano, Italy

joint work with Diego Calvanese, Benjamin Cogrel, and Guohui Xiao
MongoDB is a document database system. It is very popular and stores JSON-like documents. It offers powerful \textit{ad hoc} query languages.
Example: JSON document
From a collection (of documents) about distinguished computer scientists

{ 
  "_id": 4,
  "awards": [
    { "award": "Rosing Prize", "year": 1999 },
    { "award": "Turing Award", "by": "ACM", "year": 2001 },
    { "award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE" }
  ],
  "birth": "1926-08-27",
  "contribs": ["OOP", "Simula"],
  "death": "2002-08-10",
  "name": { "first": "Kristen", "last": "Nygaard" }
}
Example: JSON document
From a collection (of documents) about distinguished computer scientists

```
{
"_id": 4,

  "awards": [
    {
      "award": "Rosing Prize",
      "year": 1999
    },
    {
      "award": "Turing Award",
      "by": "ACM",
      "year": 2001
    },
    {
      "award": "IEEE John von Neumann Medal",
      "by": "IEEE",
      "year": 2001
    }
  ],

  "birth": "1926-08-27",

  "contribs": ["OOP", "Simula"],

  "death": "2002-08-10",

  "name": {
    "first": "Kristen",
    "last": "Nygaard"
  }
}
```
Example: JSON document
From a collection (of documents) about distinguished computer scientists

```json
{ "_id": 4,
  "awards": [
    { "award": "Rosing Prize", "year": 1999 },
    { "award": "Turing Award", "by": "ACM", "year": 2001 },
    { "award": "IEEE John von Neumann Medal", "by": "IEEE", "year": 2001 }
  ],
  "birth": "1926-08-27",
  "contribs": ["OOP", "Simula"],
  "death": "2002-08-10",
  "name": { "first": "Kristen", "last": "Nygaard" }
}
```
Example: JSON document
From a collection (of documents) about distinguished computer scientists

```json

{ "_id": 4,

  "awards": [
    { "award": "Rosing Prize", "year": 1999},
    { "award": "Turing Award", "by": "ACM", "year": 2001},
    { "award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE" } ],

  "birth": "1926-08-27",

  "contribs": ["OOP", "Simula"],

  "death": "2002-08-10",

  "name": { "first": "Kristen", "last": "Nygaard"}
}
```
Example: JSON document
From a collection (of documents) about distinguished computer scientists

```json
{ 
    "_id": 4,
    "awards": [ 
        { 
            "award": "Rosing Prize", 
            "year": 1999
        },
        { 
            "award": "Turing Award", 
            "by": "ACM", 
            "year": 2001
        },
        { 
            "award": "IEEE John von Neumann Medal", 
            "year": 2001, 
            "by": "IEEE" 
        }
    ],
    "birth": "1926-08-27",
    "contribs": ["OOP", "Simula"],
    "death": "2002-08-10",
    "name": { 
        "first": "Kristen", 
        "last": "Nygaard"
    }
}
```
Example: JSON document
From a collection (of documents) about distinguished computer scientists

```
{ "_id": 4,
  "awards": [ {"award": "Rosing Prize", "year": 1999},
               {"award": "Turing Award", "by": "ACM", "year": 2001},
               {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"} ],
  "birth": "1926-08-27",
  "contribs": ["OOP", "Simula"],
  "death": "2002-08-10",
  "name": {"first": "Kristen", "last": "Nygaard"}
}
```
Example: Find query

db.bios.find(
    {$and: [
        {
            "awards.year": {$eq: 1999}
        },
        {
            "name.first": {$eq: "Kristen"}
        }
    ],
    {
        "name": true, "birth": true
    }
)
Example: Find query

```javascript
db.bios.find({
  $and: [
    {
      "awards.year": {$eq: 1999}
    },
    {
      "name.first": {$eq: "Kristen"}
    }
  ],
  "name": true, "birth": true
})
```

When evaluated over the document about Kristen Nygaard:

```
{
  "_id": 4,
  "birth": "1926-08-27",
  "name": { "first": "Kristen", "last": "Nygaard" }
}
```
Example: Aggregation Framework query

Retrieves scientists who received two awards in the same year.

```javascript
db.bios.aggregate(
    [{
        $project: {
            "name": true,
            "award1": "$awards",
            "award2": "$awards"
        },
        $unwind: "$award1",
        $unwind: "$award2",
        $project: {
            "name": true,
            "award1": true,
            "award2": true,
            "twoInOneYear": {
                $and: [
                    {
                        $eq: [
                            "$award1.year",
                            "$award2.year"
                        ]},
                    {
                        $ne: [
                            "$award1.award",
                            "$award2.award"
                        ]}
                ]
            }
        },
        $match: {
            "twoInOneYear": true
        }
    }
)
```
Example: Aggregation Framework query

Retrieves scientists who received two awards in the same year.

db.bios.aggregate(
  [
    { $project: { "name": true,
                  "award1": "$awards", "award2": "$awards" } },
    { $unwind: "$award1" },
    { $unwind: "$award2" },
    { $project: { "name": true, "award1": true, "award2": true,
                  "twoInOneYear": { $and: [
                      { $eq: ["$award1.year", "$award2.year"]},
                      { $ne: ["$award1.award", "$award2.award"]}
                  ] } } },
    { $match: { "twoInOneYear": true } }
  ])

When evaluated over the document about Kristen Nygaard:

{ "_id": 4,
  "name": {"first": "Kristen", "last": "Nygaard"}
  "award1": {"award": "Turing Award", "by": "ACM", "year": 2001},
  "award2": {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"},
  "twoInOneYear": true }
Example: Aggregation Framework query

Retrieves scientists who received two awards in the same year.

db.bios.aggregate([
  {$project: { "name": true, 
              "award1": "$awards", "award2": "$awards" } },
  {$unwind: "$award1"},
  {$unwind: "$award2"},
  {$project: { "name": true, "award1": true, "award2": true, 
               "twoInOneYear": { $and: [ 
                {$eq: ["$award1.year", "$award2.year"]},
                {$ne: ["$award1.award", "$award2.award"]} 
              ] } }},
  {$match: { "twoInOneYear": true } },
])

When evaluated over the document about Kristen Nygaard:

{ "_id": 4,  
  "name": {"first": "Kristen", "last": "Nygaard"}  
  "award1": {"award": "Turing Award", "by": "ACM", "year": 2001},  
  "award2": {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"},  
  "twoInOneYear": true }

This query performs a **join within a document**.
Example: Another Aggregation Framework query

Retrieves pairs of scientists who received the same award the same year.

db.bios.aggregate([  
  {  
    $unwind: "$awards"},  
  {  
    $project: {  
      "awards": 1,  
      "doc._id": "$_id",  
      "doc.name": "$name"  
    }  
  },  
  {  
    $group: {  
      _id: {  
        "awardYear": "$awards.year",  
        "awardName": "$awards.award"  
      },  
      "docs": {  
        $addToSet: "$doc"  
      }  
    }  
  },  
  {  
    $project: {  
      "doc1": "$docs",  
      "doc2": "$docs"  
    }  
  },  
  {  
    $unwind: "$doc1"},  
  {  
    $unwind: "$doc2"},  
  {  
    $project: {  
      "name1": "$doc1.name",  
      "name2": "$doc2.name",  
      "awardName": "$_id.awardName",  
      "awardYear": "$_id.awardYear",  
      "toJoin": {  
        $ne: ["$_id._id", "$_id._id"]  
      }  
    }  
  },  
  {  
    $match: {"toJoin": true}  
  }])

This query performs a join across documents.
Example: Another Aggregation Framework query

Retrieves pairs of scientists who received the same award the same year.

db.bios.aggregate([  
  {"$unwind": "$awards"},  
  {"$project": {  
    "awards": 1,  
    "doc._id": "$_id",  
    "doc.name": "$name"  
  }},  
  {"$group": {  
    "_id": {  
      "awardYear": "$awards.year",  
      "awardName": "$awards.award"  
    },  
    "docs": {"$addToSet": "$doc"}  
  }},  
  {"$project": {  
    "doc1": "$docs",  
    "doc2": "$docs"  
  }},  
  {"$unwind": "$doc1"},  
  {"$unwind": "$doc2"},  
  {"$project": {  
    "name1": "$doc1.name",  
    "name2": "$doc2.name",  
    "awardName": "$_id.awardName",  
    "awardYear": "$_id.awardYear",  
    "toJoin": {"$ne": ["$_id._id", "$_id._id"]}  
  }},  
  {"$match": {"toJoin": true}}])

This query performs a join across documents.
Our Contributions

1. formalised the JSON data model
2. formalised a fragment of the aggregation framework query language ⇒ MQuery
3. analysed the expressivity and complexity of MQuery
Formalisation of the data model

Document: finite unordered, unranked, node- and edge-labeled tree
Collection: a forest of unique trees (primary key)
Formalisation of the data model

Document: finite unordered, unranked, node- and edge-labeled tree
Collection: a forest of unique trees (primary key)

Simplifying assumptions (set semantics)

- No order between
  - documents in the collection
  - key-value pairs
  - values in an array
- Multiplicity of values in an array is ignored
MongoDB aggregation framework: MQuery

- A query is a multi-stage **pipeline** applied to a collection
- A stage is a forest transformation operator
  - **match**: selects trees according to a Boolean criterion
  - **unwind**: flattens arrays at a given path
  - **project**: modifies trees by renaming, introducing, or removing paths
  - **group**: combines different trees, may create arrays
  - **lookup**: joins input trees with trees in an external collection
MongoDB aggregation framework: MQuery

- A query is a multi-stage **pipeline** applied to collection
- A stage is a forest transformation operator
  - **match** selects trees according to a Boolean criterion
  - **unwind** flattens arrays at a given path
  - **project** modifies trees by renaming, introducing, or removing paths
  - **group** combines different trees, may create arrays
  - **lookup** joins input trees with trees in an external collection

We formalised a fragment of this language as MQuery, or $\mathcal{M}^{\text{MUPGL}}$. 
Match operator: $\mu_\varphi$
Selects trees according the criterion $\varphi$

Query 1

```javascript
db.bios.aggregate([  
   {$match: {"name.first": {$eq: "Kristen"}}}
])
```

$\text{bios} \triangleright \mu \text{name.first="Kristen"}$

**input = output**

```javascript
{
   "_id": 4,
   "awards": [  
      {"award": "Rosing Prize", "year": 1999},
      {"award": "Turing Award", "by": "ACM", "year": 2001},
      {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"},
      {"award": "OOP", "Simula"},
   "birth": "1926-08-27",
   "death": "2002-08-10",
   "name": {"first": "Kristen", "last": "Nygaard"}
}
```
Match operator: $\mu_\varphi$
Selects trees according the criterion $\varphi$

**Query 2**

```javascript
db.bios.aggregate([{
  $match: {
    "awards.year": {
      $eq: 1999
    }
  }
}])
```

$\text{bios} \, \triangleright \, \mu_{\text{awards.year=1999}}$

**input = output**

```json
{
  "_id": 4,
  "awards": [
    {
      "award": "Rosing Prize",
      "year": 1999
    },
    {
      "award": "Turing Award",
      "by": "ACM",
      "year": 2001
    },
    {
      "award": "IEEE John von Neumann Medal",
      "by": "IEEE",
      "year": 2001
    }
  ],
  "birth": "1926-08-27",
  "contribs": ["OOP", "Simula"],
  "death": "2002-08-10",
  "name": {
    "first": "Kristen",
    "last": "Nygaard"
  }
}```
Match operator: $\mu_\varphi$
Selects trees according to the criterion $\varphi$

**Query 3**

db.bios.aggregate([  
  {$match: {"awards": {$eq: {"award": "Rosing Prize", "year": 1999}}}}  
])

bios ⪫ $\mu_{awards={"award": "Rosing Prize", "year": 1999}}$

**input = output**

```
{  
  "_id": 4,  
  "awards": [  
    {"award": "Rosing Prize", "year": 1999},  
    {"award": "Turing Award", "by": "ACM", "year": 2001},  
    {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"}  
  ],  
  "birth": "1926-08-27",  
  "contribs": ["OOP", "Simula"],  
  "death": "2002-08-10",  
  "name": {"first": "Kristen", "last": "Nygaard"}
}
```
Match operator: $\mu_\varphi$
Selects trees according the criterion $\varphi$

Query 4

```javascript
db.bios.aggregate([
    { $match: { "awards": { $eq: { "year": 1999, "award": "Rosing Prize" } } } }
])

bios $\triangleright \mu_{awards={"year":1999,"award":"Rosing Prize"}}$

Filtered out by the implementation but kept with our semantics

```javascript
{
    "_id": 4,
    "awards": [
        {"award": "Rosing Prize", "year": 1999},
        {"award": "Turing Award", "by": "ACM", "year": 2001},
        {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"} ],
    "birth": "1926-08-27",
    "contribs": ["OOP", "Simula"],
    "death": "2002-08-10",
    "name": {"first": "Kristen", "last": "Nygaard"}
}
```
Unwind operator: $\omega_p$
Flattens arrays at a given path $p$

**Query 1**

```javascript
db.bios.aggregate([
   {$unwind: "$awards"}
])
```

**Input**

```json
{
   "_id": 4,
   "awards": [
      {
         "award": "Rosing Prize", "year": 1999,
         "award": "Turing Award", "by": "ACM", "year": 2001,
         "award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"
      }
   ],
   "birth": "1926-08-27",
   "contribs": ["OOP", "Simula"],
   "death": "2002-08-10",
   "name": {"first": "Kristen", "last": "Nygaard"}
}
```
Unwind operator: $\omega_p$
Flattens arrays at a given path $p$

Query 1

```javascript
db.bios.aggregate([
  {$unwind: "$awards"}
])
```

Output

```json
{
  "_id": 4,
  "awards": {
    "award": "Rosing Prize", "year": 1999,
    "birth": "1926-08-27",
    "contribs": ["OOP", "Simula"],
    "death": "2002-08-10",
    "name": {
      "first": "Kristen", "last": "Nygaard"
    }
  }
{
  "_id": 4,
  "awards": {
    "award": "Turing Award", "by": "ACM", "year": 2001,
    "birth": "1926-08-27",
    "contribs": ["OOP", "Simula"],
    "death": "2002-08-10",
    "name": {
      "first": "Kristen", "last": "Nygaard"
    }
  }
  ...
```
Unwind operator: $\omega_p$
Flattens arrays at a given path $p$

Query 2

```javascript
db.bios.aggregate([{$unwind: "$publications"}])
```

bios $\bowtie \omega_{publications}$

Input

```json
{
  "_id": 4,
  "awards": [
    {"award": "Rosing Prize", "year": 1999},
    {"award": "Turing Award", "by": "ACM", "year": 2001},
    {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"}
  ],
  "birth": "1926-08-27",
  "contribs": ["OOP", "Simula"],
  "death": "2002-08-10",
  "name": {"first": "Kristen", "last": "Nygaard"}
}
```

Output

Empty
Project operator: $\rho_{p/d,...}$
Projects path $p$ according to its definition $d$

**Query 1**

```javascript
db.bios.aggregate([
    {$project: { "awards": true,
      "awardNames": "$awards.award",
      "firstName": "$name.first" }}
])
```

$\rho_{awards, awardNames/awards.award, firstName/\text{name.first}}$

**Input**

```javascript
{ "_id": 4,
  "awards": [ {"award": "Rosing Prize", "year": 1999},
    {"award": "Turing Award", "by": "ACM", "year": 2001},
    {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"} ],
  "birth": "1926-08-27",
  "contribs": ["OOP", "Simula"],
  "death": "2002-08-10",
  "name": {"first": "Kristen", "last": "Nygaard"}
}
```
Project operator: $\rho_{p/d,...}$
Projects path $p$ according to its definition $d$

**Query 1**

```javascript
db.bios.aggregate([
  {$project: {
    "awards": true,
    "awardNames": "$awards.award",
    "firstName": "$name.first"
  }}
])
```

Output

```
{
  "_id": 4,
  "awards": [ {
    "award": "Rosing Prize", "year": 1999},
    {
    "award": "Turing Award", "by": "ACM", "year": 2001},
    {
    "award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"} ],
  "awardNames": [ "Rosing Prize", "Turing Award", "IEEE John von Neumann Medal" ],
  "firstName": "Kristen"
}
```
Project operator: $\rho_{p/d,...}$
Projects path $p$ according to its definition $d$

**Query 2**

```
db.bios.aggregate([
    {$project: { "calledJohn": {$eq: ["$name.first", "John"]},
       "sameFirstAndLastNames": {$eq: ["$name.first", "$name.last"]},
       "newArray": ["$name.first", "$name.last"],
       "condValue": {cond: { if: {$eq: ["$_id", 4]},
                          then: "$name.first",
                          else: "$awards" }}},
    "invisible": "$abc" }]
])
```

```
bios \supset \rho_{calledJohn/(name.first="John"), sameFirstAndLastNames/(name.first=name.last),
        newArray/[name.first,name.last], condValue/(_id=4?name.first:awards)
```

**Input**

```
{ "_id": 4,  
  "awards": [ {"award": "Rosing Prize", "year": 1999},
              {"award": "Turing Award", "by": "ACM", "year": 2001},
              {"award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE"} ],
  "birth": "1926-08-27",
  "contribs": ["OOP", "Simula"],
  "death": "2002-08-10",
  "name": {"first": "Kristen", "last": "Nygaard"}
}
```
Project operator: $\rho_{p/d},...$
Projects path $p$ according to its definition $d$

**Query 2**

```javascript
db.bios.aggregate([
    {$project: {
        "calledJohn": {$eq: ['$name.first', 'John']},
        "sameFirstAndLastNames": {$eq: ['$name.first', '$name.last']},
        "newArray": ['$name.first', '$name.last'],
        "condValue": {
            "cond": {
                if: {$eq: ['$_id', 4]},
                then: "$name.first",
                else: "$awards"
            }
        },
        "invisible": "$abc"
    }}
])
```

`bios ⊳ ρ_{calledJohn/(name.first="John"), sameFirstAndLastNames/(name.first=name.last), newArray/[name.first,name.last], condValue/(_id=4?name.first:awards)}`

**Output**

```javascript
{ "_id": 4,
  "calledJohn": false,
  "sameFirstAndLastNames": false,
  "newArray": [ "Kristen", "Nygaard" ],
  "condValue": "Kristen"
}
```
Group operator: $\gamma_G:A$
Groups trees according to $G$ and collects values according to $A$

**Query**

```javascript
db.bios.aggregate([
    {$unwind: "$awards"},
    {$group: { "_id": { "year": "$awards.year" },
                  "names": {$addToSet: "$name" } }},
])
```

\[ \text{bios } \bowtie \omega \text{awards } \bowtie \gamma \text{year/awards.year}\text{:names/name} \]

**Input**

```
{ "_id": 4,
  "awards": [ { "award": "Rosing Prize", "year": 1999 },
                { "award": "Turing Award", "year": 2001 },
                { "award": "IEEE John von Neumann Medal", "year": 2001 } ],
  "name": { "first": "Kristen", "last": "Nygaard" } }

{ "_id": 6,
  "awards": [ { "award": "Award for the Advancement of Free Software", "year": 2001 },
                { "award": "NLUUG Award", "year": 2003 } ],
  "name": { "first": "Guido", "last": "van Rossum" } }
```
Group operator: $\gamma_{G:A}$
Groups trees according to $G$ and collects values according to $A$

Query

```javascript
db.bios.aggregate([
  {$unwind: "$awards"},
  {$group: { "_id": {"year": "$awards.year"},
                "names": {$addToSet: "$name"} }},
])
```

Output

```json
{ "_id": { "year": 2003 },
  "names": [ { "first": "Guido", "last": "van Rossum" } ] },

{ "_id": { "year": 2001 },
  "names": [ { "first": "Kristen", "last": "Nygaard" },
                { "first": "Guido", "last": "van Rossum" } ] },

{ "_id": { "year": 1999 },
  "names": [ { "first": "Kristen", "last": "Nygaard" } ] }
```
Lookup operator: $\lambda_p^{p_1=C.p_2}$
Performs left outer join to the collection $C$ and stores joined documents under $p$. 

**Query**

```javascript
db.bios.aggregate([
  {$unwind: "$awards"},
  {$group: {_id: {"year": "$awards.year"}, "names": {$addToSet: "$name"} }}],
  {$lookup: { from: "Events", localField: "_id.year",
    foreignField: "year",
    as: "joinedDocs" }}]
)
```

**bios**

```json
{ 
  "_id": 4,
  "awards": [ 
    { "award": "Rosing Prize", "year": 1999 },
    { "award": "Turing Award", "year": 2001 },
    { "award": "IEEE John von Neumann Medal", "year": 1999 },
    "name": { "first": "Kristen", "last": "Nygaard" }
  ]
}

{ 
  "_id": 6,
  "awards": [ 
    { "award": "Award for the Advancement of Free Software", "year": 2001 },
    { "award": "NLUUG Award", "year": 2003 } ],
    "name": { "first": "Guido", "last": "van Rossum" }
  ]
}
```

**Events**

```json
{ 
  "_id": 1,
  "year": 1997,
  "event": "Deep Blue defeats Garry Kasparov"
}

{ 
  "_id": 2,
  "year": 1999,
  "event": "Melissa virus outbreak"
}

{ 
  "_id": 3,
  "year": 1999,
  "event": "Jeff Bezos is person of the year"
}
```
Lookup operator: $\lambda_p^{p_1=C.p_2}$
Periods left outer join to the collection $C$ and stores joined documents under $p$

Query

```javascript
db.bios.aggregate([
    {$unwind: "$awards"},
    {$group: {_id: {"year": "$awards.year"}, "names": {$addToSet: "$name"} }},
    {$lookup: { from: "Events", localField: "_id.year",
                foreignField: "year",
                as: "joinedDocs" }}
])
```

Output

```
{
    "_id": { "year": 2003 },
    "names": [ { "first": "Guido", "last": "van Rossum" } ],
    "joinedDocs": []
},
{
    "_id": { "year": 2001 },
    "names": [ { "first": "Kristen", "last": "Nygaard" },
               { "first": "Guido", "last": "van Rossum" } ]
    "joinedDocs": []
},
{
    "_id": { "year": 1999 },
    "names": [ { "first": "Kristen", "last": "Nygaard" } ]
    "joinedDocs": [ { "_id": 2, "year": 1999, "event": "Melissa virus outbreak" },
                     { "_id": 3, "year": 1999, "event": "Jeff Bezos is person of the year" } ]
}
```
Expressivity of MQuery

Characterized in terms of Nested Relational Algebra (NRA)

1. Nested relational view of JSON documents
2. Translation from NRA to MQuery
3. Translation from MQuery to NRA
## Nested Relational View

<table>
<thead>
<tr>
<th>id</th>
<th>awards</th>
<th>birth</th>
<th>contribs</th>
<th>death</th>
<th>name.first</th>
<th>name.last</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Rosing Prize</td>
<td>1926-08-27</td>
<td>OOP</td>
<td>2002-08-10</td>
<td>Kristen</td>
<td>Nygaard</td>
</tr>
<tr>
<td></td>
<td>Turing Award</td>
<td>1999</td>
<td>Simula</td>
<td>2001</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IEEE John von Neumann Medal</td>
<td>2001</td>
<td></td>
<td>2002-08-10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Only possible for well-typed forests**

- Each path is typed
- Analogous to complex object types and JSON schema
Nested Relational Algebra (NRA)
Recap.

- **Relational Algebra operators:**
  - Selection
  - Extended projection
  - Cross-product
  - Union
  - Minus

- **Unnest:** flattens a nested sub-relation

<table>
<thead>
<tr>
<th>_id</th>
<th>awards</th>
<th>name.first</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>award</td>
<td>year</td>
</tr>
<tr>
<td>4</td>
<td>Rosing Prize</td>
<td>1999</td>
</tr>
<tr>
<td></td>
<td>Turing Award</td>
<td>2001</td>
</tr>
<tr>
<td></td>
<td>IEEE John von...</td>
<td>2001</td>
</tr>
</tbody>
</table>

\[ \chi_{awards} \]

<table>
<thead>
<tr>
<th>_id</th>
<th>award</th>
<th>year</th>
<th>name.first</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Rosing Prize</td>
<td>1999</td>
<td>Kristen</td>
</tr>
<tr>
<td>4</td>
<td>Turing Award</td>
<td>2001</td>
<td>Kristen</td>
</tr>
<tr>
<td>4</td>
<td>IEEE John von...</td>
<td>2001</td>
<td>Kristen</td>
</tr>
</tbody>
</table>

- **Nest:** creates nested sub-relation

<table>
<thead>
<tr>
<th>_id</th>
<th>award</th>
<th>year</th>
<th>name.first</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Rosing Prize</td>
<td>1999</td>
<td>Kristen</td>
</tr>
<tr>
<td>4</td>
<td>Turing Award</td>
<td>2001</td>
<td>Kristen</td>
</tr>
<tr>
<td>4</td>
<td>IEEE John von...</td>
<td>2001</td>
<td>Kristen</td>
</tr>
</tbody>
</table>

\[ \nu\{award\} \rightarrow awards \]

<table>
<thead>
<tr>
<th>_id</th>
<th>awards</th>
<th>year</th>
<th>name.first</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>award</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rosing Prize</td>
<td>1999</td>
<td>Kristen</td>
</tr>
<tr>
<td></td>
<td>Turing Award</td>
<td>2001</td>
<td>Kristen</td>
</tr>
<tr>
<td></td>
<td>IEEE John von...</td>
<td>2001</td>
<td>Kristen</td>
</tr>
</tbody>
</table>

\[ \nu\{award\} \rightarrow awards \]

Elena Botoeva(FUB)  
Expressivity and Complexity of MongoDB queries  
17/22
Compact Translation from NRA to MQuery

Expressivity
- $\mathcal{M}^{\text{MUPGL}}$ captures NRA
- $\mathcal{M}^{\text{MUPG}}$ captures NRA over a single collection

Main technical challenge
“Linearize” a tree-shaped NRA expression into a MongoDB pipeline
Compact Translation from NRA to MQuery

**Expressivity**

- MQuery ($\mathcal{M}^{MUPGL}$) captures NRA
- $\mathcal{M}^{MUPG}$ captures NRA over a single collection

**Main technical challenge**

“Linearize” a tree-shaped NRA expression into a MongoDB pipeline

For two MQueries $q_1$ and $q_2$, we construct a pipeline that does the following:

$$\{\} \quad \{\} \quad \{\} \quad \{\} \quad \{\} \quad \{\} \quad \{\} \quad \{\} \quad \{\} \quad \{\}$$

$$1 \quad 1 \quad 2 \quad 2$$

$$actRel \ rel1 \quad actRel \ rel1 \quad actRel \ rel2 \quad actRel \ rel2$$

$$t_1 \in (F \triangleright q_1) \quad t_2 \in (F \triangleright q_2)$$
Compact Translation from MQuery to NRA

Expressivity
Well-typed MQuery is captured by NRA

- Stages that transform well-typed forests into well-typed forests

- match $\Rightarrow$ selection
- unwind $\Rightarrow$ unnest
- project $\Rightarrow$ projection
- group $\Rightarrow$ nest
- lookup $\Rightarrow$ left outer join

Challenges
MQuery stages can “look” inside arrays
## Complexity of MQuery

Data complexity: $AC^0$

<table>
<thead>
<tr>
<th>Fragment</th>
<th>Query complexity</th>
<th>Combined complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{M}^M$</td>
<td>LOGSPACE-complete</td>
<td>PTIME-complete</td>
</tr>
<tr>
<td>$\mathcal{M}^{MP}, \mathcal{M}^{MPGL}$</td>
<td>LOGSPACE-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>$\mathcal{M}^{MU}$</td>
<td>LOGSPACE-complete</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{M}^{MUP}, \mathcal{M}^{MUL}, \mathcal{M}^{MUPL}$</td>
<td>NP-complete</td>
<td>PSPACE-hard</td>
</tr>
<tr>
<td>$\mathcal{M}^{MUG}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{M}^{MUPG}, \mathcal{M}^{MUPGL}$</td>
<td>TA$[2^{n^{O(1)}}, n^{O(1)}]$-complete*</td>
<td></td>
</tr>
<tr>
<td>NRA</td>
<td>TA$[2^{n^{O(1)}}, n^{O(1)}]$-complete</td>
<td></td>
</tr>
</tbody>
</table>

* The class of problems solvable by an alternating Turing machine running in exponential time with polynomially many alternations.
Concluding remarks

Technical report
http://arxiv.org/abs/1603.09291

(Expected) Outcomes
- Enable the integration of MongoDB within the OBDA framework
- Could influence the evolution of MongoDB

Future work
- Relaxed notion of well-typedness
- Bag and list semantics
- New operators (e.g. graph-lookup)
See you at the poster!