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

a document database system 

- Very popular
- Stores JSON-like documents
- Offers powerful **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", "year": 2001, "by": "IEEE"} ],

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

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

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

  "name": {"first": "Kristen", "last": "Nygaard"}
}
```

Keys

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"}
}
```

Values

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"}
}
```

Literals

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"}
}
```

Nested Objects

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"}
}
```

Arrays

Example: Find query

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

Example: Find query

```
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.

```
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": {$ne: ["$doc1._id", "$doc2._id"]} }},
    {$match: {"toJoin": true}}
])
```

Example: Another Aggregation Framework query

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

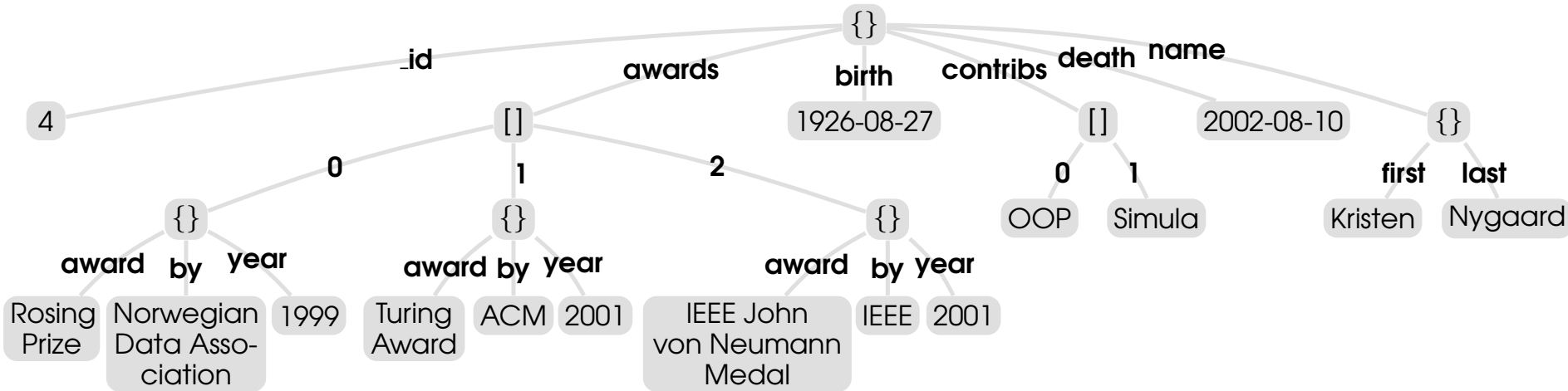
```
db.bios.aggregate([
    {$unwind: "$awards"},
    {$project: { "awards": 1,
                 "doc._id": "$_id",
                 "doc.name": "$name" }},
    {$group: { _id: { "awardYear": "$awards.year",
                      "awardName": "$awards.award" },
               "docs": {$addToSet: "$doc"} }},
    {$project: { "doc1": "$docs",
                  "doc2": "$docs" }},
    {$unwind: "$doc1"},
    {$unwind: "$doc2"},
    {$project: { "name1": "$doc1.name",
                 "name2": "$doc2.name",
                 "awardName": "$_id.awardName",
                 "awardYear": "$_id.awardYear",
                 "toJoin": {$ne: ["$doc1._id", "$doc2._id"]} }},
    {$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 $\Rightarrow$ MQuery
3. analysed the **expressivity** and **complexity** of MQuery

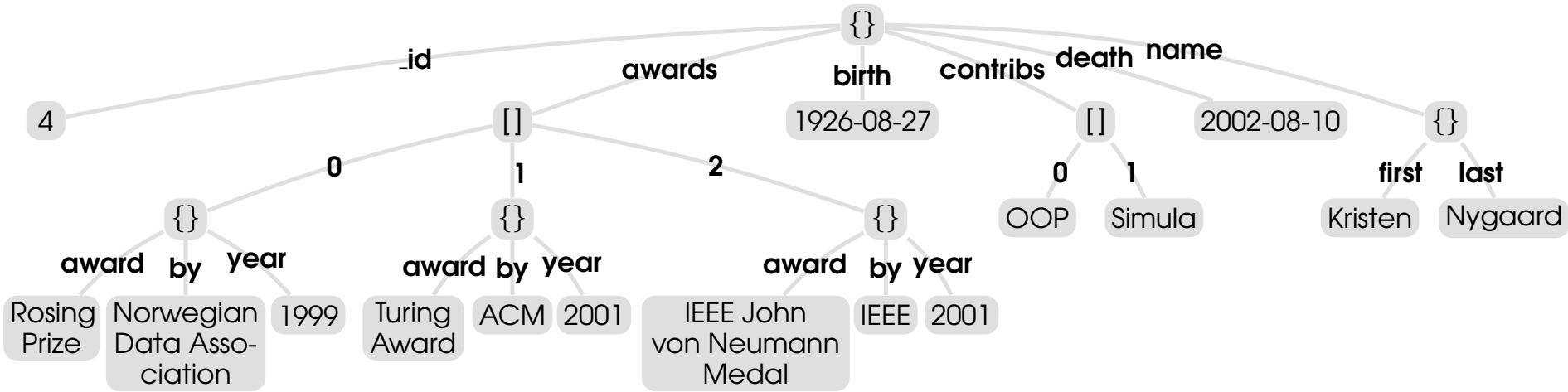
Formalisation of the data model



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 collection
- A stage is a forest transformation operator
  - **m**atch selects trees according to a Boolean criterion
  - **u**nwind flattens arrays at a given path
  - **p**roject modifies trees by renaming, introducing, or removing paths
  - **g**roup combines different trees, may create arrays
  - **l**ookup 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
  - **m**atch selects trees according to a Boolean criterion
  - **u**nwind flattens arrays at a given path
  - **p**roject modifies trees by renaming, introducing, or removing paths
  - **g**roup combines different trees, may create arrays
  - **l**ookup 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: μ_φ

Selects trees according the criterion φ

Query 1

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

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

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: μ_φ

Selects trees according the criterion φ

Query 2

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

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

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: μ_φ

Selects trees according the criterion φ

Query 3

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

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

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: μ_φ

Selects trees according the criterion φ

Query 4

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

`bios` $\triangleright$ $\mu_{\text{awards=}\{\text{“year”: 1999, “award”: “Rosing Prize”}\}}$

Filtered out by the implementation but kept with our semantics

```
{ "_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: ω_p

Flattens arrays at a given path p

Query 1

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

bios $\triangleright\ \omega_{\text{awards}}$

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"}
}
```

Unwind operator: ω_p

Flattens arrays at a given path p

Query 1

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

bios $\triangleright\ \omega_{\text{awards}}$

Output


```
{ "_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: ω_p

Flattens arrays at a given path p

Query 2

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

bios $\triangleright$ $\omega_{\text{publications}}$

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"}
}
```

Output

Empty

Project operator: $\rho_{p/d,\ldots}$

Projects path p according to its definition d

Query 1

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

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

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: $\rho_{p/d,\ldots}$

Projects path p according to its definition d

Query 1

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

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

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: $\rho_{p/d,\ldots}$

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 $\triangleright$ $\rho_{\text{calledJohn}/(\text{name.first}=\text{“John”}),\ \text{sameFirstAndLastNames}/(\text{name.first}=\text{name.last}),\ \text{newArray}/[\text{name.first},\text{name.last}],\ \text{condValue}/(\text{\_id}=4?\text{name.first}:\text{awards})$

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: $\rho_{p/d,\ldots}$

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` $\triangleright$ $\rho_{\text{calledJohn}/(\text{name.first}=\text{“John”}),\ \text{sameFirstAndLastNames}/(\text{name.first}=\text{name.last}),\ \text{newArray}/[\text{name.first},\text{name.last}],\ \text{condValue}/(\text{\_id}=4?\text{name.first}:\text{awards})}$

Output

```
{ "_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

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

bios $\triangleright$ ω_{awards} $\triangleright$ $\gamma_{\text{year/awards.year:names/name}}$

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

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

`bios` $\triangleright$ ω_{awards} $\triangleright$ $\gamma_{\text{year/awards.year:names/name}}$

Output

```
{ "_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

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

$\texttt{bios} \triangleright \omega_{\text{awards}} \triangleright \gamma_{\text{year/awards.year:names/name}} \triangleright \lambda_{\text{joinedDocs}}^{\text{\_id.year=Events.year}}$

bios

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

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

Events

```
{  "_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: $\lambda_p^{p_1=C.p_2}$

Performs left outer join to the collection C and stores joined documents under p

Query

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

$\texttt{bios} \triangleright \omega_{\text{awards}} \triangleright \gamma_{\text{year/awards.year:names/name}} \triangleright \lambda_{\text{joinedDocs}}^{\text{\_id.year=Events.year}}$

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

_id	awards		birth	contribs	death	name.first	name.last
	award	year		lit			
4	Rosing Prize	1999	1926-08-27	OOP	2002-08-10	Kristen	Nygaard
	Turing Award	2001		Simula			
	IEEE John von Neumann Medal	2001					

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

_id	awards		name.first
	award	year	
4	Rosing Prize	1999	Kristen
	Turing Award	2001	
	IEEE John von...	2001	

$\xrightarrow{\chi_{\text{awards}}}$

_id	award	year	name.first
4	Rosing Prize	1999	Kristen
4	Turing Award	2001	Kristen
4	IEEE John von...	2001	Kristen

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

_id	award	year	name.first
4	Rosing Prize	1999	Kristen
4	Turing Award	2001	Kristen
4	IEEE John von...	2001	Kristen

$\xrightarrow{\nu_{\{\text{award}\}\rightarrow\text{awards}}}$

_id	awards	year	name.first
	award		
4	Rosing Prize	1999	Kristen
4	Turing Award	2001	Kristen
	IEEE John von...		

Compact Translation from NRA to MQuery

Expressivity

- MQuery ($\mathcal{M}^{\text{MUPGL}}$) 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

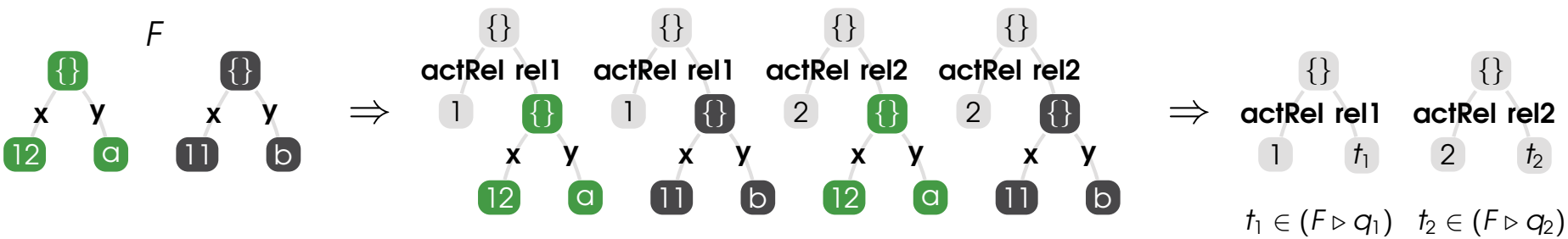
Expressivity

- MQuery ($\mathcal{M}^{\text{MUPGL}}$) 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

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



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

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

$*$ 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!