Query languages for property graphs

From RPQs to Cypher

NoSQL and New SQL course

M2 LID, Université Gustave-Eiffel

2023 - 2024

version 2

Introduction

About this PDF

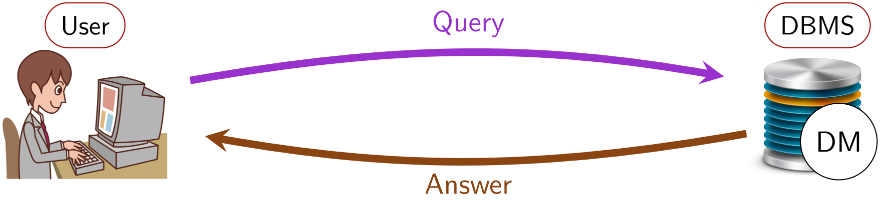
Navigation

From any frame, the page number is a link to the navigable outline

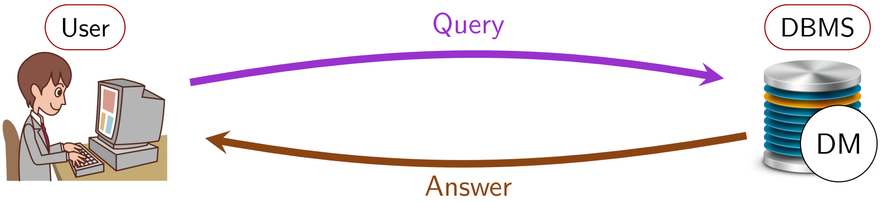
Term translations

There is a French/English lexicon at the end.

Overview of query answering

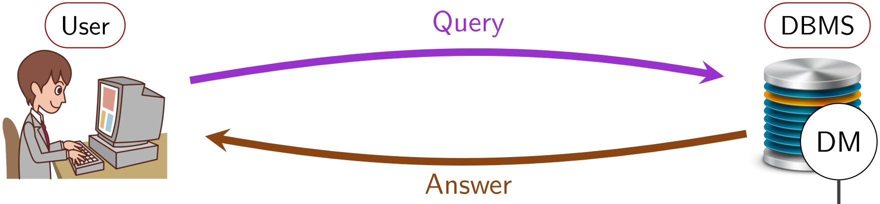


- **DBMS** (DataBase Management System)



Overview of query answering

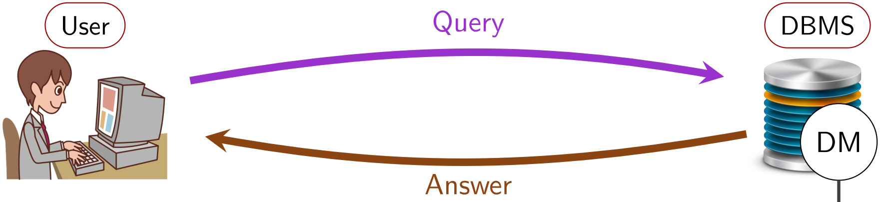
- **DBMS** (DataBase Management System)



- **DM** (**Data Model**)
  - *"How is the data structured?" "What data is representable?"*
  - Ex: Relations (SQL), Trees (XML, JSON), Graphs (PGs, RDF),

Overview of query answering

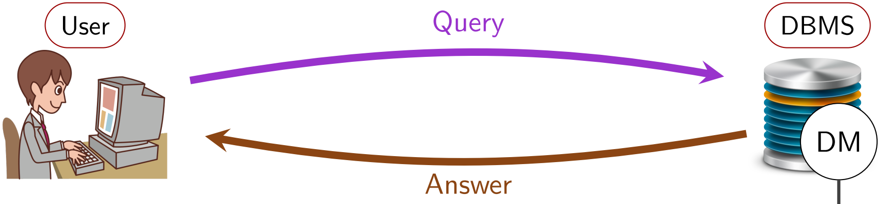
- **DBMS** (DataBase Management System)
- **Query language**
  - *"What can the user write?"*



- **DM** (**Data Model**)
  - *"How is the data structured?"* *"What data is representable?"*
  - Ex: Relations (SQL), Trees (XML, JSON), Graphs (PGs, RDF),

Overview of query answering

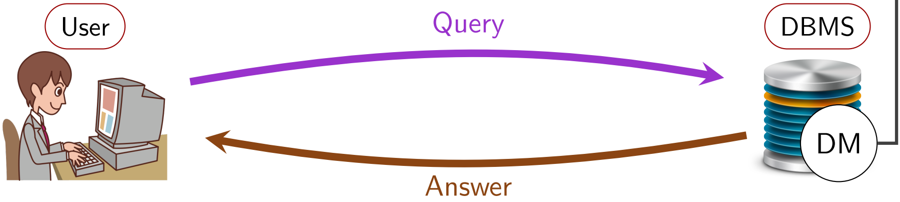
- **DBMS** (DataBase Management System)
- **Query language**
  - *"What can the user write?"*



- **Semantics**
  - *"What does the query mean?" "What is the correct answer?"*
  - Ex: Set semantics (duplicate elimination)
- **DM** (**Data Model**)
  - *"How is the data structured?" "What data is representable?"*
  - Ex: Relations (SQL), Trees (XML, JSON), Graphs (PGs, RDF),

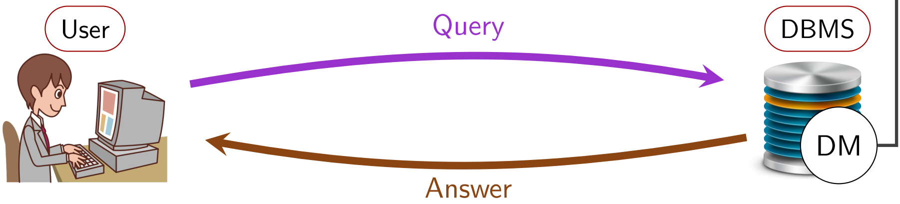
This segment is about **query languages for property graphs**

- The **data model** is **Property Graph (PG)**



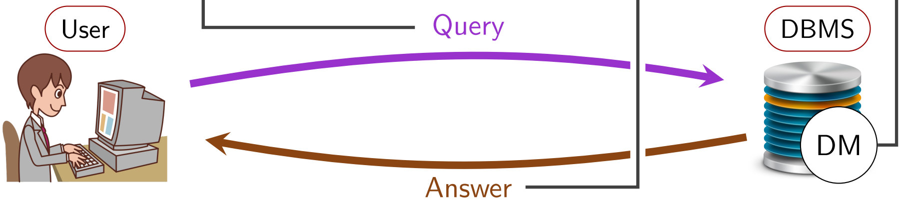
This segment is about **query languages for property graphs**

- The **data model** is **Property Graph (PG)**
- The **DBMS** we will use (**Neo4j**) implements this DM



This segment is about **query languages for property graphs**

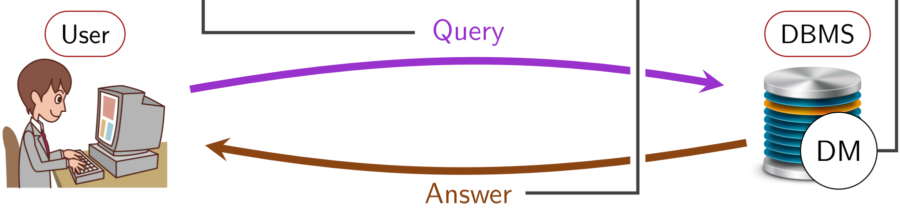
- The **data model** is **Property Graph (PG)**
- The **DBMS** we will use (**Neo4j**) implements this DM
- The **query languages** we consider (**Cypher**, GQL, etc.) need this DM to have **any meaning**



This segment is about **query languages for property graphs**

In part **II**:

- The **data model** is **Property Graph (PG)**
- The **DBMS** we will use (**Neo4j**) implements this DM
- The **query languages** we consider (**Cypher**, GQL, etc.) need this DM to have **any meaning**



Popularity of Graph DBMS's (1)

Vast majority of DMBS's are relational, not graph

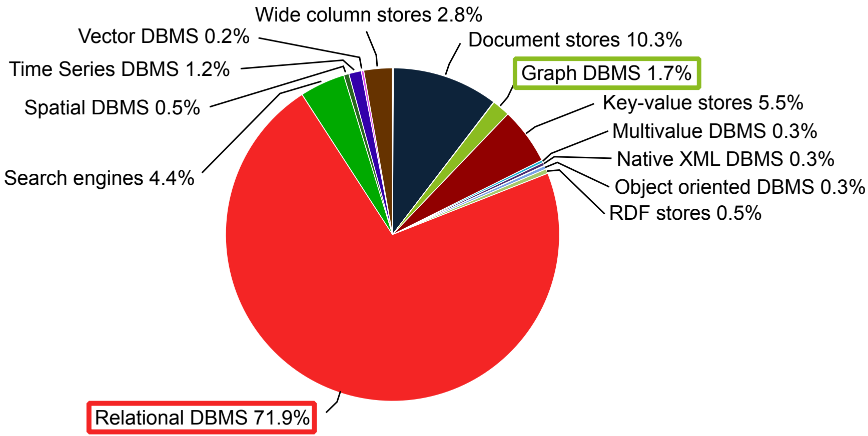


Figure and data from `db-engines.com`, August 2023

Popularity of Graph DBMS's (2)

Graph DBMS's has grown in popularity for ten years

Relational DBMS's continued their slow decline

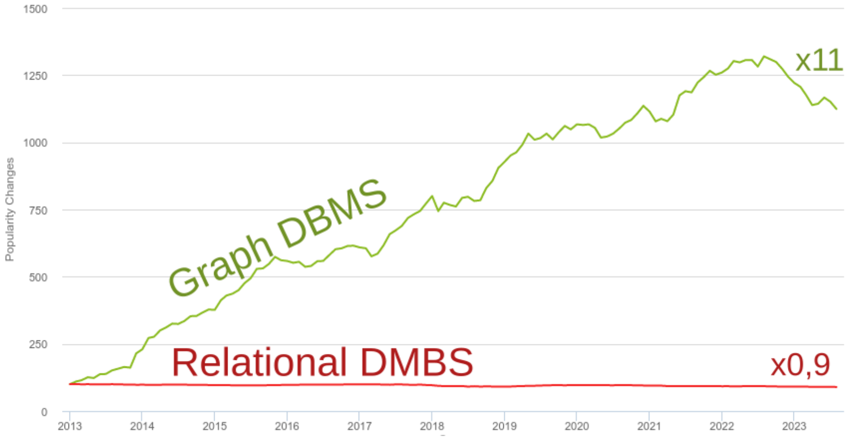
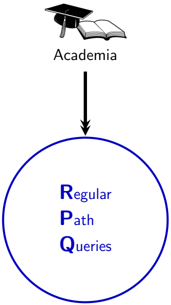


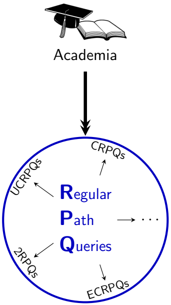
Figure and data from `db-engines.com`, August 2023

A bit of history



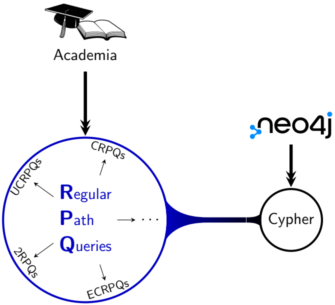
Late 1980's – RPQs are invented

A bit of history



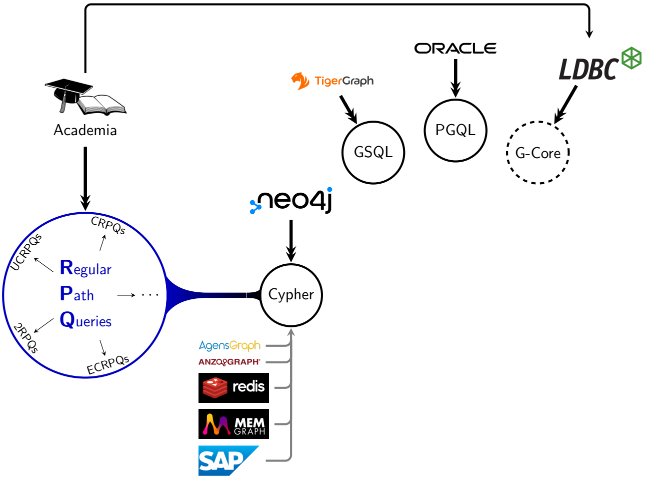
Since 1990's – RPQs are studied and extended in academia

A bit of history



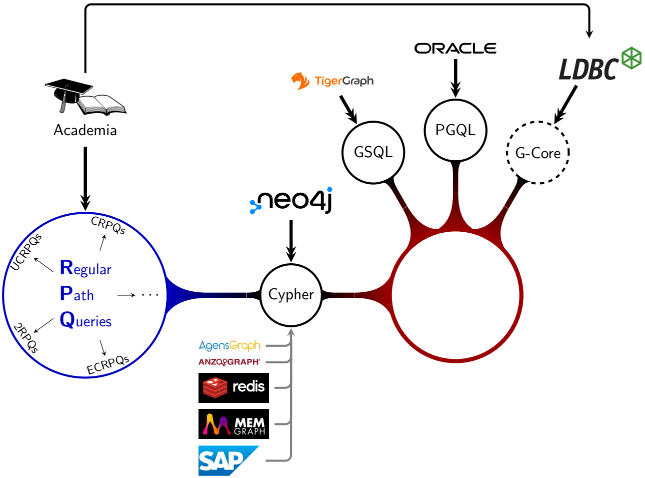
2011 – The query language Cypher is released with the DBMS Neo4j

A bit of history



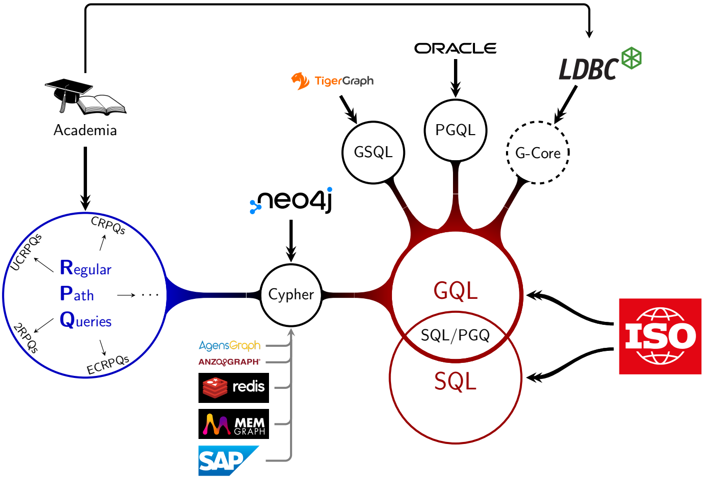
Mid 2010's – Cypher is successful and new graph DBMS's appear. Some use Cypher, some come with their own query language.

A bit of history



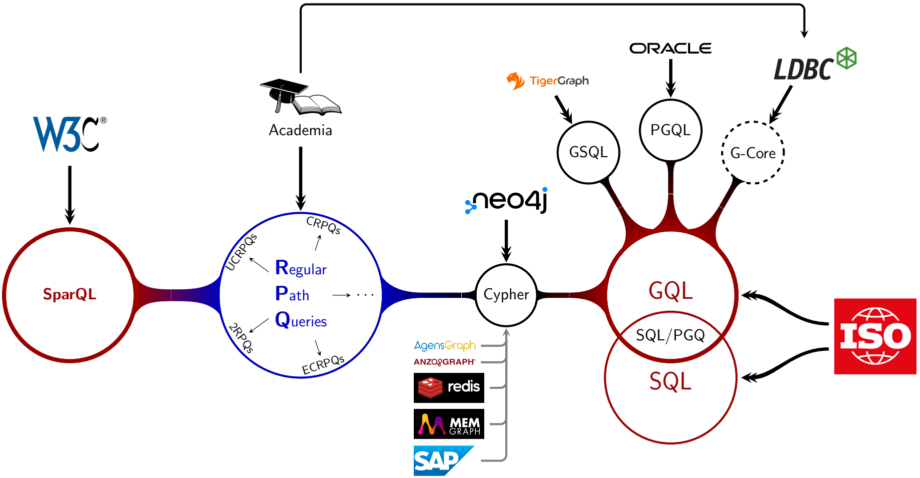
Late 2010's – Idea to merge existing languages for interoperability

A bit of history



2023 – SQL/PGQ support for querying PG's in SQL
2024? – GQL, standard query language for PG's

A bit of history



Side note: In SPARQL, the standard language for the RDF DM, features *Property paths* which are also based on RPQ's.

Outline

Part I: Theoretical Foundations

- Data model: Graphs
- Query language: RPQs

Part II: A practical application

- Data model: Property graphs
- Query language: Cypher

Part I: Theoretical foundations

Terminology reminder: sets

A **set** is a finite or infinite collection of elements such that:

- order does not matter
- duplicates do not matter

Example sets:

- $\{1\} = \{1, 1\}$
- $\{4, 8, 15, 16, 23, 42\}$
  $= \{8, 4, 23, 15, 42, 16\}$
- $\{(4, 15), (16, 42)\}$
- $\mathbb{N} = \{0, 1, 2, \ldots\}$
- $\emptyset$, the empty set

Terminology reminder: sets

A **set** is a finite or infinite collection of elements such that:

- order does not matter
- duplicates do not matter

Example sets:

- $\{1\} = \{1, 1\}$
- $\{4, 8, 15, 16, 23, 42\}$
  $= \{8, 4, 23, 15, 42, 16\}$
- $\{(4, 15), (16, 42)\}$
- $\mathbb{N} = \{0, 1, 2, \ldots\}$
- $\emptyset$, the empty set

The **union** of two sets A and B is the set of elements which are in A, in B, or in both A and B.

$\{1, 3\} \cup \{2, 1\} = \{1, 3, 2\}$

Terminology reminder: sets

A **set** is a finite or infinite collection of elements such that:

- order does not matter
- duplicates do not matter

Example sets:

- $\{1\} = \{1, 1\}$
- $\{4, 8, 15, 16, 23, 42\}$
  $= \{8, 4, 23, 15, 42, 16\}$
- $\{(4, 15), (16, 42)\}$
- $\mathbb{N} = \{0, 1, 2, \ldots\}$
- $\emptyset$, the empty set

The **union** of two sets A and B is the set of elements which are in A, in B, or in both A and B.

$\{1, 3\} \cup \{2, 1\} = \{1, 3, 2\}$

The **intersection** of two sets A and B is the set of elements which are in both A and B.

$\{1, 3\} \cap \{2, 1\} = \{1\}$

Terminology reminder: sets

A **set** is a finite or infinite collection of elements such that:

- order does not matter
- duplicates do not matter

Example sets:

- $\{1\} = \{1, 1\}$
- $\{4, 8, 15, 16, 23, 42\} = \{8, 4, 23, 15, 42, 16\}$
- $\{(4, 15), (16, 42)\}$
- $\mathbb{N} = \{0, 1, 2, \ldots\}$
- $\emptyset$, the empty set

The **union** of two sets A and B is the set of elements which are in A, in B, or in both A and B.

$\{1, 3\} \cup \{2, 1\} = \{1, 3, 2\}$

The **intersection** of two sets A and B is the set of elements which are in both A and B.

$\{1, 3\} \cap \{2, 1\} = \{1\}$

The **Cartesian product** of two sets A and B is the set of all pairs (x, y) for $x \in A$ and $y \in B$

$$\{1, 2, 3\} \times \{a, b\} = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$$

Part I: Theoretical foundations

1. Data model: labeled graphs

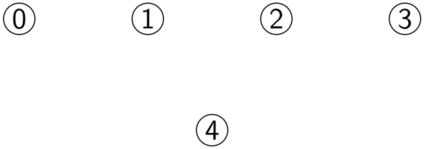
A graph consists of ...

- Vertices
- Edges
- Edge labels

Our data model : (Labeled) graphs (1)

A graph consists of ...

- Vertices
- Edges
- Edge labels

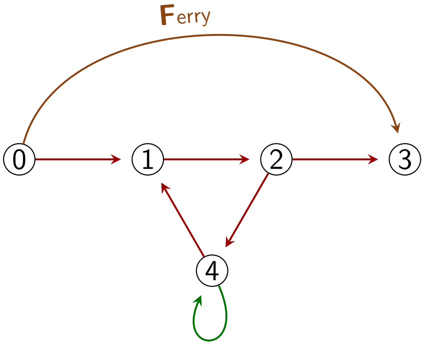


Our data model : (Labeled) graphs (1)

Example

A graph consists of ...

- Vertices
- Edges
- Edge labels

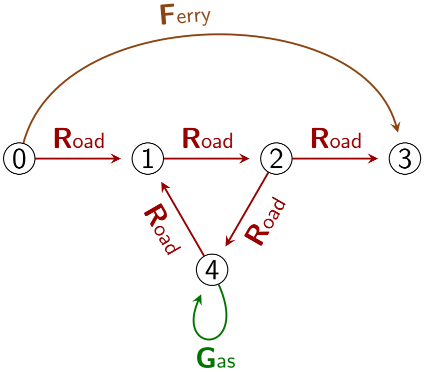


Our data model : (Labeled) graphs (1)

Example

A graph consists of ...

- Vertices
- Edges
- Edge labels



Our data model : (Labeled) graphs (2)

Formalisation

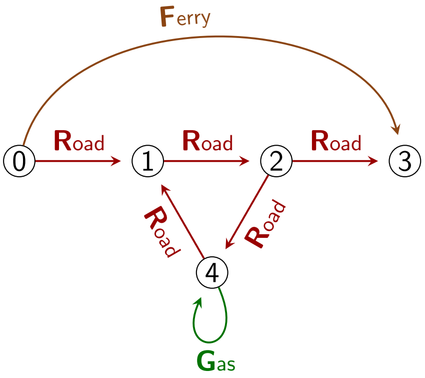
Definition

A labeled graph is a triplet (V, L, E)

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$



Example graph G

Our data model : (Labeled) graphs (2)

Formalisation

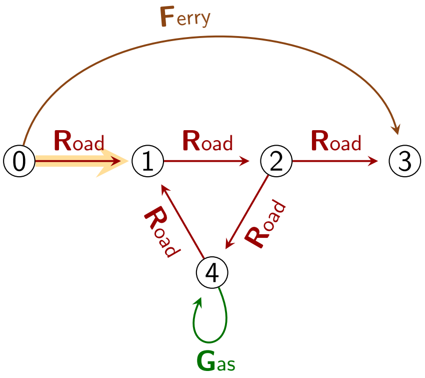
Definition

A labeled graph is a triplet (V, L, E)

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$



Example graph G

Our data model : (Labeled) graphs (2)

Formalisation

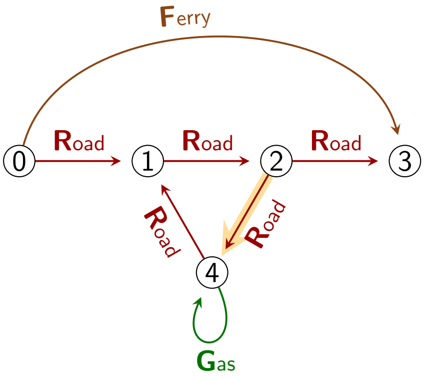
Definition

A labeled graph is a triplet (V, L, E)

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$



Example graph G

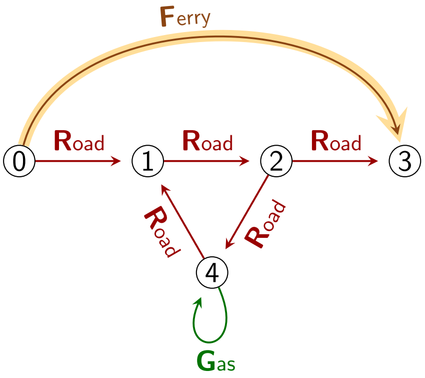
Definition

A labeled graph is a triplet (V, L, E)

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$



Example graph G

Our data model : (Labeled) graphs (2)

Formalisation

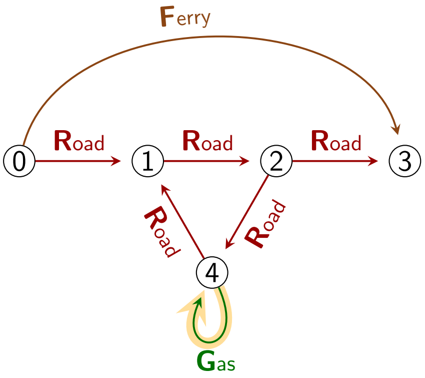
Definition

A labeled graph is a triplet (V, L, E)

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$

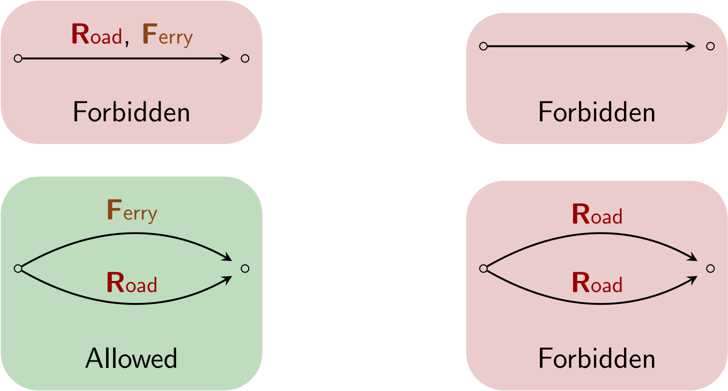


Example graph G

Limits to the graph data model (1)

Our graphs are single-labeled and single-edge

- Each edge has exactly one label.
- There cannot be two identical edges.



Limits to the graph data model (2)

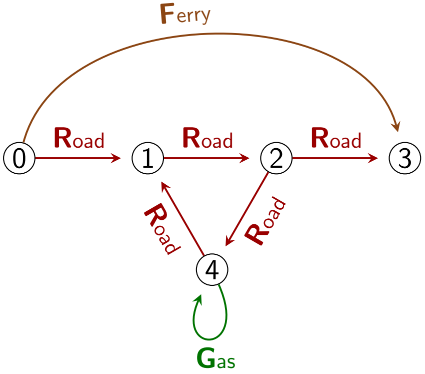
The graph DM is about topology, not data

- We encode the existence of entities and of relations between entities
  Ex: cities, roads
- We don't encode specific data of an entity or relation
  Ex: names, distances

Examples

Our model cannot encode that

- the road from 0 to 1 is 2km long
- the gas price is 2€ in vertex 4

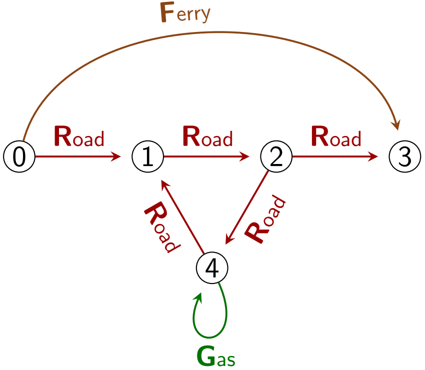


Part I: Theoretical foundations

2. Graph DM vs Relational DM

Translation: Graph to Tables (1)

Can a graph be stored in tables?



Translation: Graph to Tables (1)

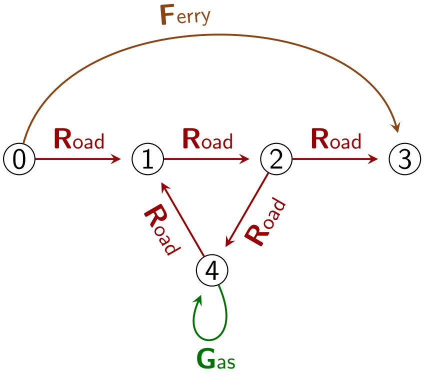
Example – One **Vertex** table with one row per vertex in the graph

Vertex

<u>id</u>
0
1
2
3
4

Road

#<u>src</u>	#<u>tgt</u>
0	1
1	2
2	3
2	4
4	1



Ferry

#<u>src</u>	#<u>tgt</u>
0	3

Gas

#<u>src</u>	#<u>tgt</u>
4	4

Translation: Graph to Tables (1)

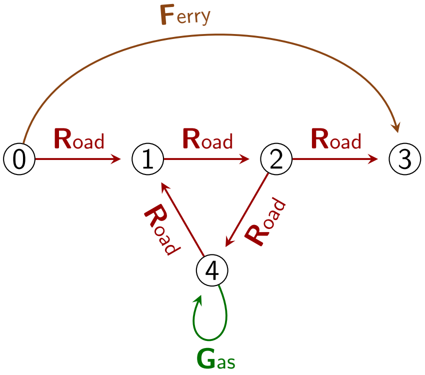
Example – One table for each different label in the graph

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (1)

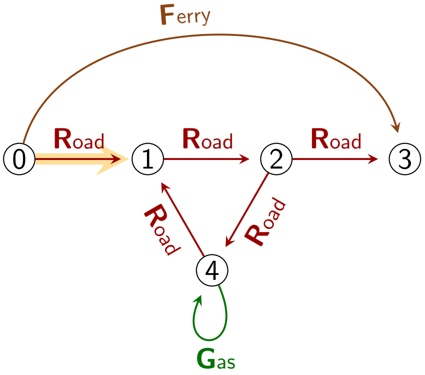
Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (1)

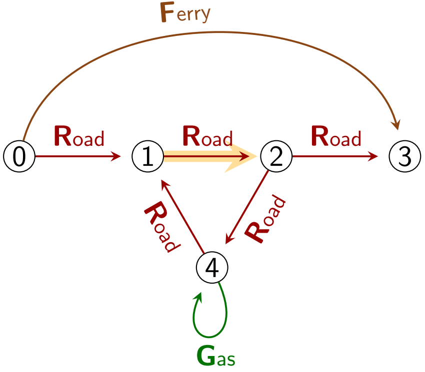
Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (1)

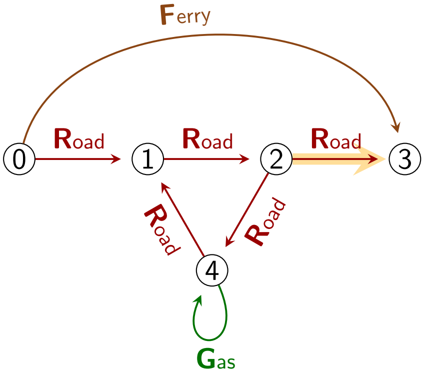
Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (1)

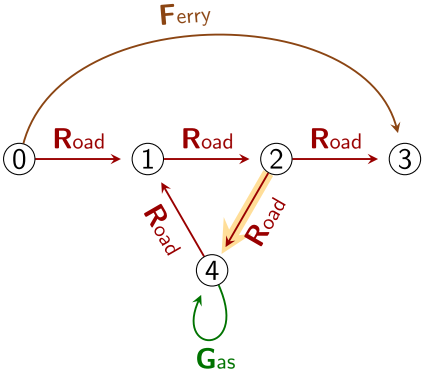
Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (1)

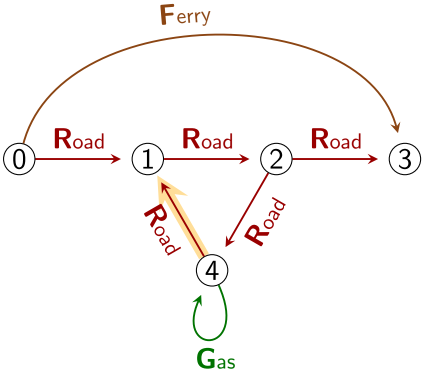
Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (1)

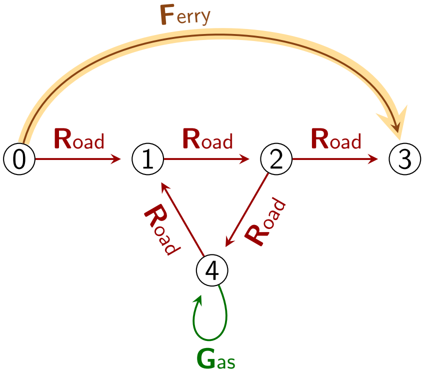
Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (1)

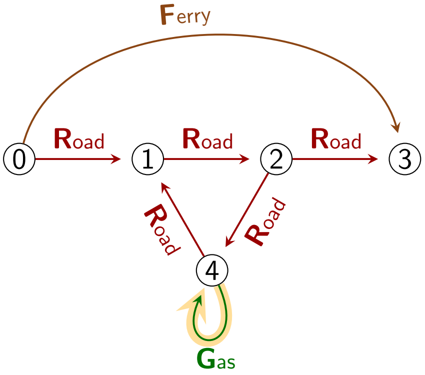
Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (2)

Principles of the translation

We start from a graph (V, L, E)
Since V is finite we may enumerate it: $V = \{v_1, \ldots, v_n\}$

One table for vertices

Vertex

<u>id</u>
0
1
⋮
n

One table per label ℓ in L

ℓ

#<u>src</u>	#<u>tgt</u>
⋮	⋮
i	j
⋮	⋮

Table ℓ contains (i, j)
$\iff (v_i, \ell, v_j) \in E$

Translation: Tables to Graph (1)

The relational database we want to encode in a graph

Client

<u>login</u>	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	"WALL-E's Truck"

Product

<u>name</u>	price
"Pocket Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

___ : part of primary key

Translation: Tables to Graph (1)

The relational database we want to encode in a graph

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	"WALL-E's Truck"

Order

id	#buyer	date
0	"Alice"	01-11-1865
1	"Bob"	07-07-2022
2	"Bob"	07-11-2023

Product

name	price
"Pocket Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

___ : part of primary key

\#   foreign keys

Translation: Tables to Graph (1)

The relational database we want to encode in a graph

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	"WALL-E's Truck"

Order

id	#buyer	date
0	"Alice"	01-11-1865
1	"Bob"	07-07-2022
2	"Bob"	07-11-2023

Product

name	price
"Pocket Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

Contains

#order	#product
0	"Rabbit"
0	"Pocket Watch"
1	"Pants"
2	"Pants"

___ : part of primary key
: foreign keys

Translation: Tables to Graph (2)

Condition for the translation to be possible

Relational DB consists of tables $T_1, \ldots, T_k$.

Each table T_i

- has a primary key, consisting of several columns
- has columns that are foreign keys

⚠ Foreign keys can be part of the primary key.

Translation: Tables to Graph (3)

Condition for the translation to be possible

Relational DB consists of tables $T_1, \ldots, T_k$.

Each table T_i

- has a primary key, consisting of several columns
- has columns that are foreign keys

⚠ Foreign keys can be part of the primary key.

Conditions for the database to be encodable in a graph

Each table T_i satisfies one of the following.

0. Zero foreign key is part of the primary key of T_i.
1. One foreign key is part of the primary key of T_i.
2. Two foreign keys are part of the primary key of T_i, and no other column is part of the primary key.

Translation: Tables to Graph (4)

The relational database we want to encode in a graph

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	"WALL-E's Truck"

Order

id	#buyer	date
0	"Alice"	01-11-1865
1	"Bob"	07-07-2022
2	"Bob"	07-11-2023

Product

name	price
"Pocket Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

Contains

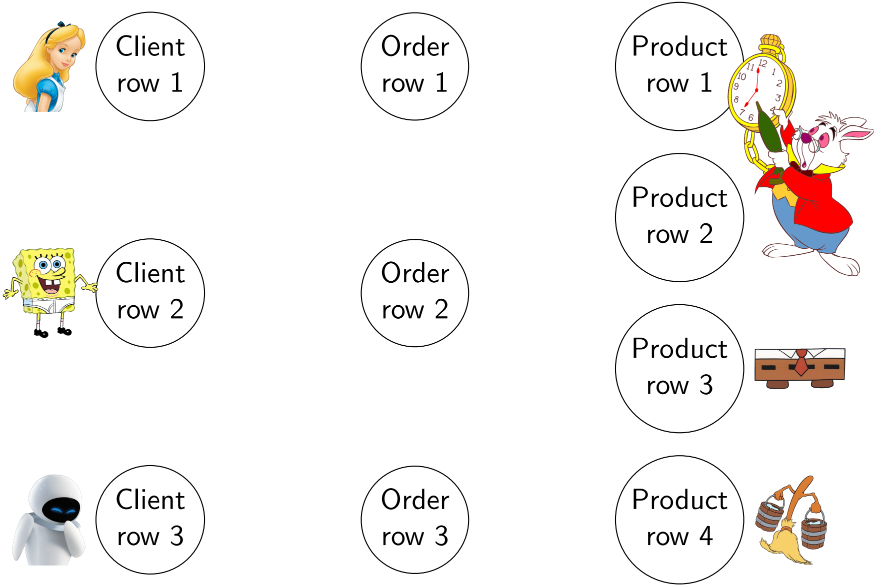
#order	#product
0	"Rabbit"
0	"Pocket Watch"
1	"Pants"
2	"Pants"

___ : part of primary key
: foreign keys

- **Client**, **Product** and **Order** satisfy 0
- **Contains** satisfies 2

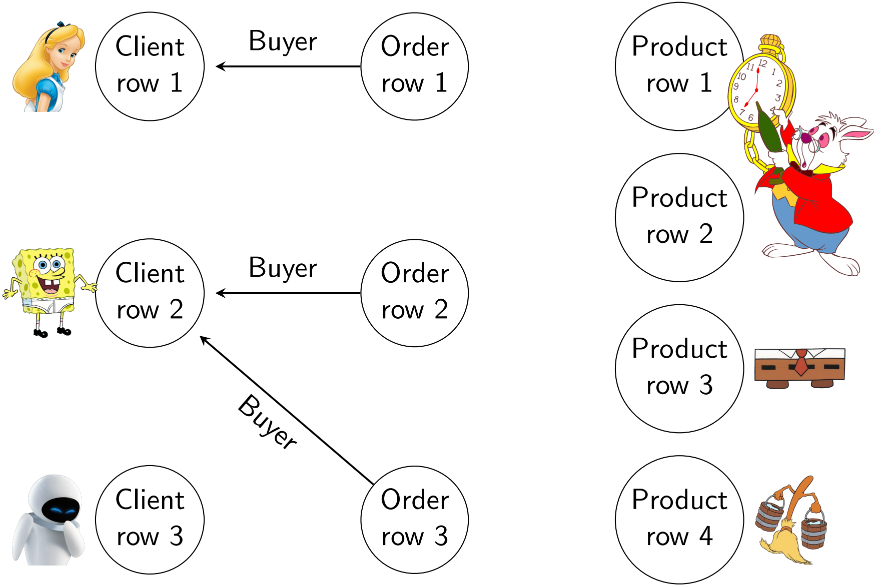
Translation: Tables to Graph (5)

One vertex per row in table satisfying 0 or 1



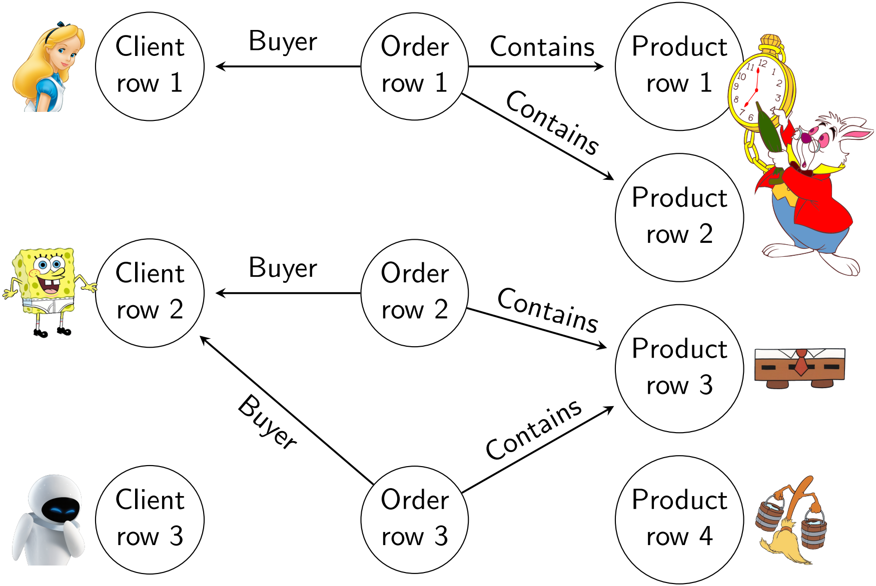
Translation: Tables to Graph (5)

One edge per row and per foreign-key column in each table satisfying 0 or 1



Translation: Tables to Graph (5)

One edge per row of tables satisfying 2



Translation: Tables to Graph (6)

Takeaway

Wedding is edgy...

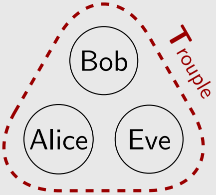
Wedding

#person1	#person2
Alice	Bob
⋮	⋮

..but troupe is trouble

Trouple

#person1	#person2	#person3
Alice	Bob	Eve
⋮	⋮	⋮



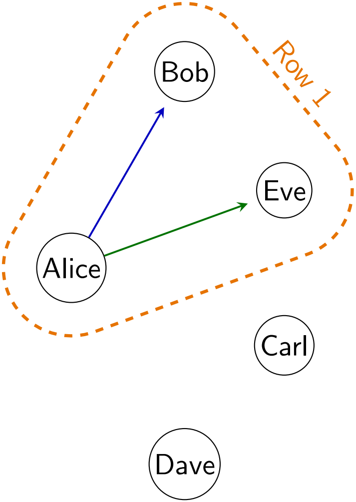
How to encode non-binary relations in a graph (1)

The **wrong** ways: adding more edges

Troupple

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

Let us try to add two edges per row of table **Troupple**.



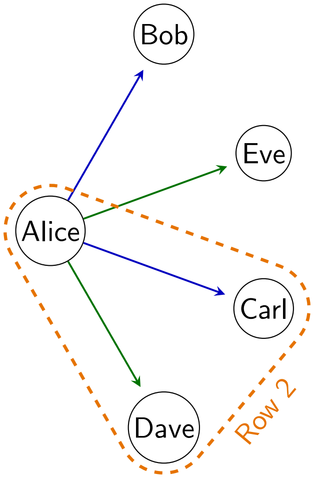
How to encode non-binary relations in a graph (1)

The **wrong** ways: adding more edges

Troupple

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

Let us try to add two edges per row of table **Troupple**.



How to encode non-binary relations in a graph (1)

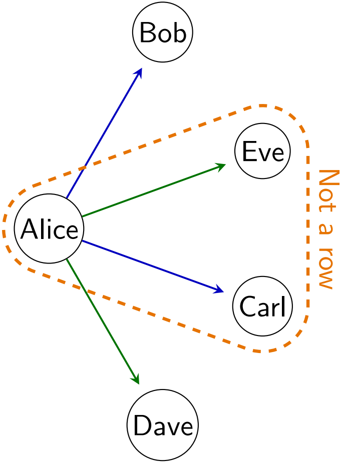
The **wrong** ways: adding more edges

Trouple

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

Let us try to add two edges per row of table **Trouple**.

⚠ (Alice, Carl, Eve) is not a row of table **Trouple**



Reification

- Literally, make into an object
- For us, transform into a vertex

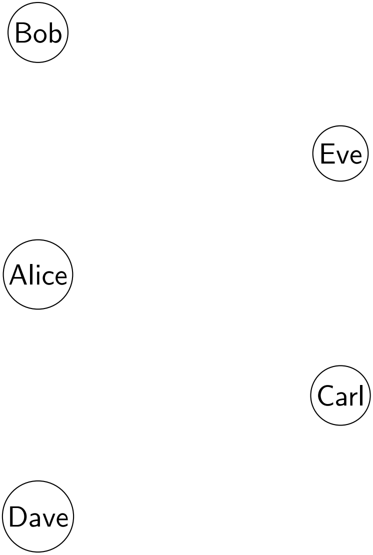
How to encode non-binary relations in a graph (2)

The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a vertex

Troupe		
#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave



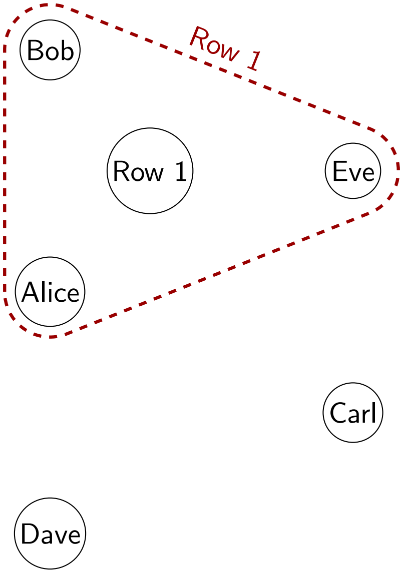
How to encode non-binary relations in a graph (2)

The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a vertex

Troupe		
#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave



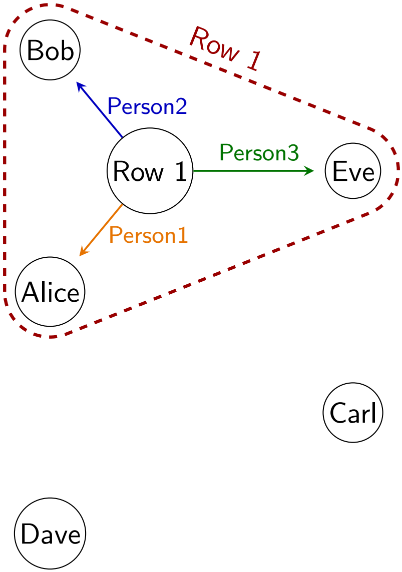
How to encode non-binary relations in a graph (2)

The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a vertex

Trouple		
#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave



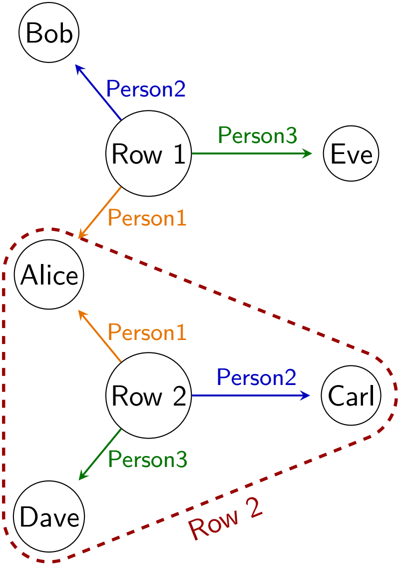
How to encode non-binary relations in a graph (2)

The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a vertex

Troupe		
#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave



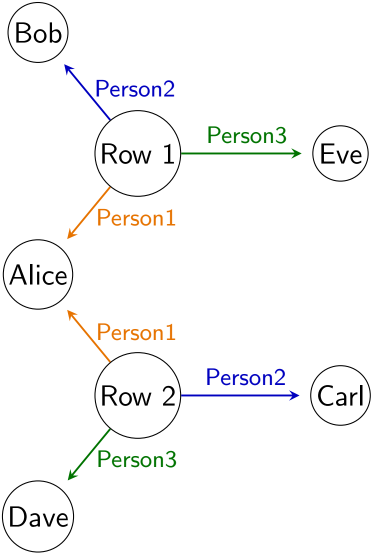
How to encode non-binary relations in a graph (2)

The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a vertex

Troupe		
#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave



Reification is **cheating**

Reification works...

- Reversible (one may reconstruct the **Trouble** table)
- Easy to generalize to any arity

...but, it is contrary to the spirit of graphs:

- The graph requires extra knowledge and maintenance:
  - Special vertices/edges/labels
  - Implictly linked labels/edges (Person1/Person2/Person3)
  - Integrity constraints
- Query languages designed for graphs will not expect them

Part I: Theoretical foundations

3. Regular Path Queries

Terminology reminder from automata theory

A **letter** is a symbol coming from a finite set, the **alphabet**.

In our case, the alphabet is the label-set of the graph.

Examples:

- $\{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$ is an alphabet
- **R** and **G** are letters

A **word** is a finite sequence of letters

Examples words:

- **RGRR**
- **R**
- ε, the empty word

A **language** is a finite or infinite set of words

Example languages:

- $\{\mathbf{R}, \mathbf{RG}\}$
- $\{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$
- The words with one **G**
- The words with a prime number of **G**

The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R**, and **F** are regexps

The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R**, and **F** are regexps

Concatenation ·

If Q_1 and Q_2 are regexps
Then $Q_1 \cdot Q_2$ is a regexp

Ex: $\mathbf{R} \cdot \mathbf{R}$ and $\mathbf{G} \cdot \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) \cdot (\mathbf{G} \cdot \mathbf{F})$ is a regexp

The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R**, and **F** are regexps

Disjunction $+$

If Q_1 and Q_2 are regexps
Then $Q_1 + Q_2$ is a regexp

Ex: $\mathbf{R} + \mathbf{R}$ and $\mathbf{G} + \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) + (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Concatenation $\cdot$

If Q_1 and Q_2 are regexps
Then $Q_1 \cdot Q_2$ is a regexp

Ex: $\mathbf{R} \cdot \mathbf{R}$ and $\mathbf{G} \cdot \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) \cdot (\mathbf{G} \cdot \mathbf{F})$ is a regexp

The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R**, and **F** are regexps

Concatenation $\cdot$

If Q_1 and Q_2 are regexps
Then $Q_1 \cdot Q_2$ is a regexp

Ex: $\mathbf{R} \cdot \mathbf{R}$ and $\mathbf{G} \cdot \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) \cdot (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Disjunction $+$

If Q_1 and Q_2 are regexps
Then $Q_1 + Q_2$ is a regexp

Ex: $\mathbf{R} + \mathbf{R}$ and $\mathbf{G} + \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) + (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Kleene star *

If Q is a regexp
Then Q^* is a regexp

Ex: $\mathbf{R}^*$ and $\mathbf{G}^*$ are regexps
$((\mathbf{R}^* \cdot \mathbf{G}) + \mathbf{F})^*$ is a regexp

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$
- $L(\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}) = \{\mathbf{RFG}\}$

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$
- $L(\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}) = \{\mathbf{RFG}\}$
- $L(\mathbf{R} + \mathbf{G}) = \{\mathbf{R}, \mathbf{G}\}$

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$
- $L(\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}) = \{\mathbf{RFG}\}$
- $L(\mathbf{R} + \mathbf{G}) = \{\mathbf{R}, \mathbf{G}\}$
- $L(\mathbf{R} \cdot \mathbf{R} + \mathbf{G} \cdot \mathbf{R}) = L((\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}) = \{\mathbf{RR}, \mathbf{GR}\}$

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$
- $L(\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}) = \{\mathbf{RFG}\}$
- $L(\mathbf{R} + \mathbf{G}) = \{\mathbf{R}, \mathbf{G}\}$
- $L(\mathbf{R} \cdot \mathbf{R} + \mathbf{G} \cdot \mathbf{R}) = L((\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}) = \{\mathbf{RR}, \mathbf{GR}\}$
- $L(\mathbf{R}^*) = \{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$
- $L(\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}) = \{\mathbf{RFG}\}$
- $L(\mathbf{R} + \mathbf{G}) = \{\mathbf{R}, \mathbf{G}\}$
- $L(\mathbf{R} \cdot \mathbf{R} + \mathbf{G} \cdot \mathbf{R}) = L((\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}) = \{\mathbf{RR}, \mathbf{GR}\}$
- $L(\mathbf{R}^*) = \{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$
- $L((\mathbf{R} + \mathbf{G})^*) = \{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \ldots\}$

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$
- $L(\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}) = \{\mathbf{RFG}\}$
- $L(\mathbf{R} + \mathbf{G}) = \{\mathbf{R}, \mathbf{G}\}$
- $L(\mathbf{R} \cdot \mathbf{R} + \mathbf{G} \cdot \mathbf{R}) = L((\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}) = \{\mathbf{RR}, \mathbf{GR}\}$
- $L(\mathbf{R}^*) = \{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$
- $L((\mathbf{R} + \mathbf{G})^*) = \{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \ldots\}$
- $L((\mathbf{R} \cdot \mathbf{R})^*) = \{\varepsilon, \mathbf{RR}, \mathbf{RRRR}, \mathbf{RRRRRR}, \ldots\}$
  "words of even length"

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$
- $L(\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}) = \{\mathbf{RFG}\}$
- $L(\mathbf{R} + \mathbf{G}) = \{\mathbf{R}, \mathbf{G}\}$
- $L(\mathbf{R} \cdot \mathbf{R} + \mathbf{G} \cdot \mathbf{R}) = L((\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}) = \{\mathbf{RR}, \mathbf{GR}\}$
- $L(\mathbf{R}^*) = \{\varepsilon,\ \mathbf{R},\ \mathbf{RR},\ \mathbf{RRR},\ \ldots\}$
- $L((\mathbf{R} + \mathbf{G})^*) = \{\varepsilon,\ \mathbf{R},\ \mathbf{G},\ \mathbf{RR},\ \mathbf{RG},\ \mathbf{GG},\ \ldots\}$
- $L((\mathbf{R} \cdot \mathbf{R})^*) = \{\varepsilon,\ \mathbf{RR},\ \mathbf{RRRR},\ \mathbf{RRRRRR},\ \ldots\}$
  "words of even length"
- $L(\mathbf{R}^* \cdot \mathbf{G} \cdot \mathbf{R}^*) = \{\mathbf{G}, \mathbf{RG}, \mathbf{GR}, \mathbf{RGR}, \mathbf{RRG}, \ldots\}$
  "words over $\{\mathbf{G}, \mathbf{R}\}$ *with exactly one* **G***"*

Language described by a regexp Q

Each regexp Q **describes** a language $L(Q)$

Examples:

- $L(\mathbf{R}) = \{\mathbf{R}\}$
- $L(\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}) = \{\mathbf{RFG}\}$
- $L(\mathbf{R} + \mathbf{G}) = \{\mathbf{R}, \mathbf{G}\}$
- $L(\mathbf{R} \cdot \mathbf{R} + \mathbf{G} \cdot \mathbf{R}) = L((\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}) = \{\mathbf{RR}, \mathbf{GR}\}$
- $L(\mathbf{R}^*) = \{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$
- $L((\mathbf{R} + \mathbf{G})^*) = \{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \ldots\}$
- $L((\mathbf{R} \cdot \mathbf{R})^*) = \{\varepsilon, \mathbf{RR}, \mathbf{RRRR}, \mathbf{RRRRRR}, \ldots\}$
  "words of even length"
- $L(\mathbf{R}^* \cdot \mathbf{G} \cdot \mathbf{R}^*) = \{\mathbf{G}, \mathbf{RG}, \mathbf{GR}, \mathbf{RGR}, \mathbf{RRG}, \ldots\}$
  "words over $\{\mathbf{G}, \mathbf{R}\}$ with exactly one $\mathbf{G}$"

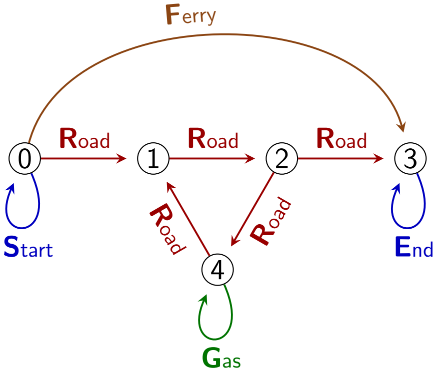
Any language described by a regexp is called **regular**

Definition of RPQs and matching walks

Definition of RPQs and matching walks

A **Regular Path Query (RPQ)**

- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$



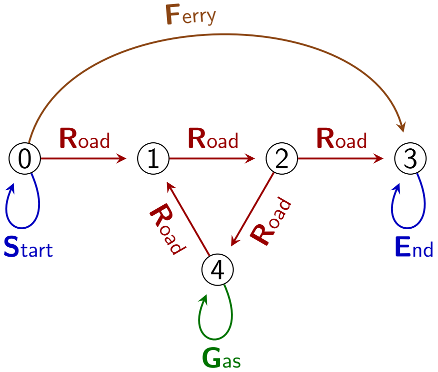
Definition of RPQs and matching walks

A **Regular Path Query (RPQ)**
- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$

A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

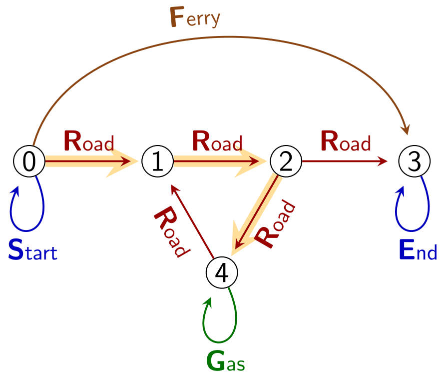
The **label of a walk** is the **word** formed by the label of its edges.

Example walk	Label
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4$	**RRR**
$0 \xrightarrow{S} 0 \xrightarrow{F} 3$	**SF**
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4 \xrightarrow{G}$ $4 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 3$	**RRRGRRR**



Definition of RPQs and matching walks

A **Regular Path Query (RPQ)**
- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$



A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.

Example walk	Label
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4$	**RRR**
$0 \xrightarrow{S} 0 \xrightarrow{F} 3$	**SF**
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4 \xrightarrow{G} 4 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 3$	**RRRGRRR**

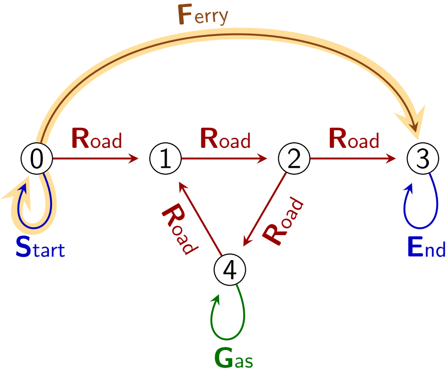
Definition of RPQs and matching walks

A **Regular Path Query (RPQ)**

- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$

A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.



Example walk	Label
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4$	**RRR**
$0 \xrightarrow{S} 0 \xrightarrow{F} 3$	**SF**
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4 \xrightarrow{G} 4 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 3$	**RRRGRRR**

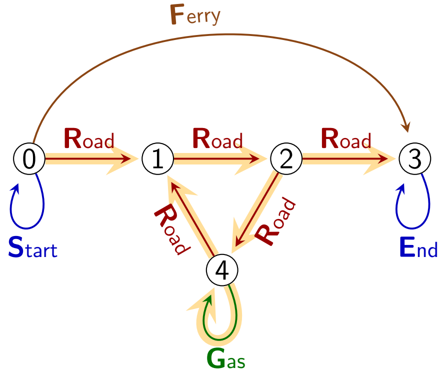
Definition of RPQs and matching walks

A **Regular Path Query (RPQ)**
- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$

A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.

Example walk	Label
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4$	RRR
$0 \xrightarrow{S} 0 \xrightarrow{F} 3$	SF
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4 \xrightarrow{G} 4 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 3$	RRRGRRR



Definition of RPQs and matching walks

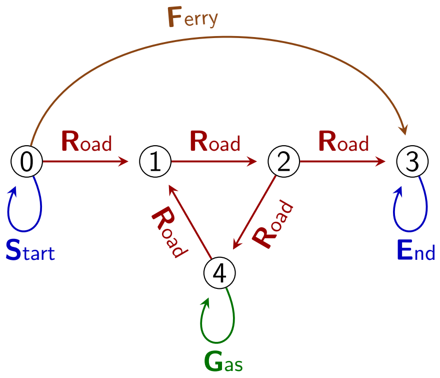
A **Regular Path Query (RPQ)**

- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$

A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.

Example walk	Label
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4$	**RRR**
$0 \xrightarrow{S} 0 \xrightarrow{F} 3$	**SF**
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4 \xrightarrow{G} 4 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 3$	**RRRGRRR**

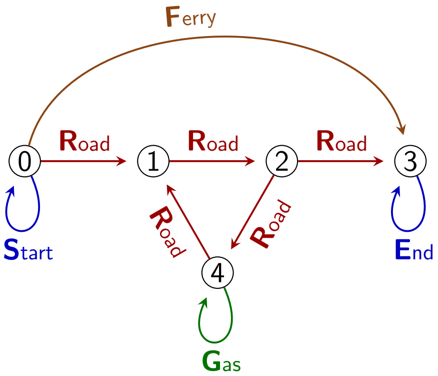


A **walk** w is a **match** to an **RPQ** Q if the **label** of w is in $L(Q)$.

Matching atoms

Matching query $Q_1 = \mathbf{R}$

$L(Q_1) = \{\mathbf{R}\}$

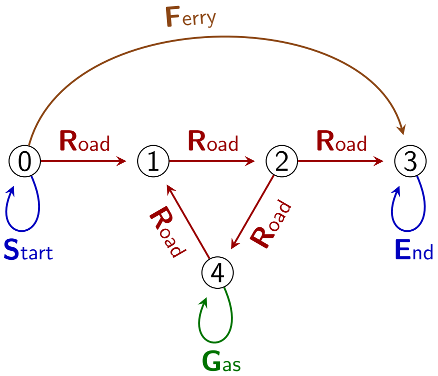


Matching atoms

Matching query $Q_1 = \mathbf{R}$

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.



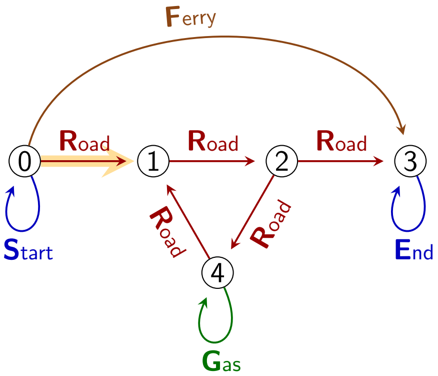
Matching atoms

Matching query $Q_1 = \mathbf{R}$

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**

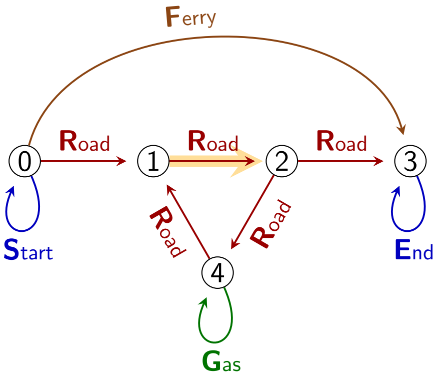


Matching query $Q_1 = \mathbf{R}$

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**

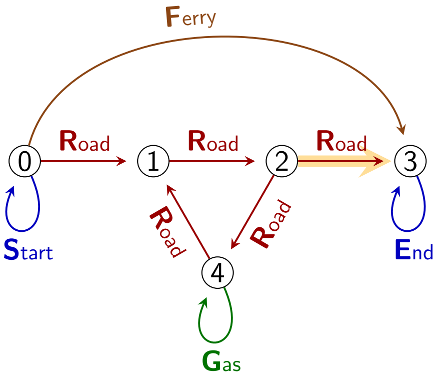


Matching query $Q_1 = \mathbf{R}$

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**



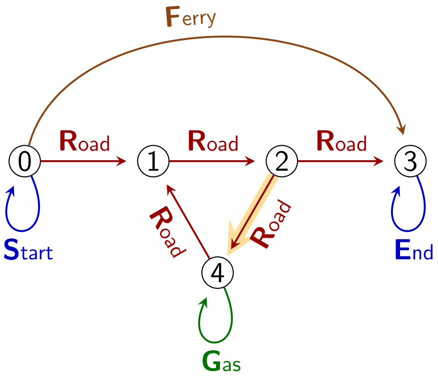
Matching atoms

Matching query $Q_1 = \mathbf{R}$

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**



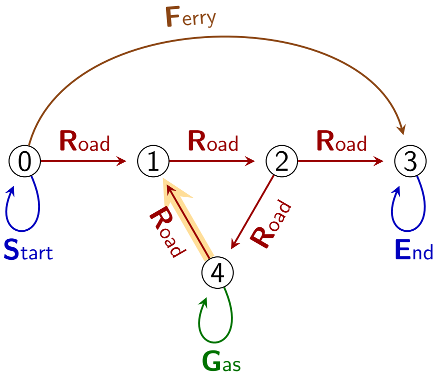
Matching atoms

Matching query $Q_1 = \mathbf{R}$

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**



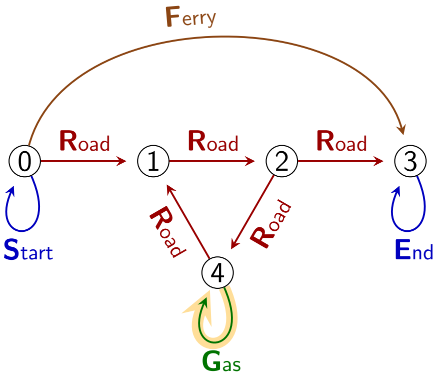
Matching atoms

Matching query $Q_1 = \mathbf{R}$

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**



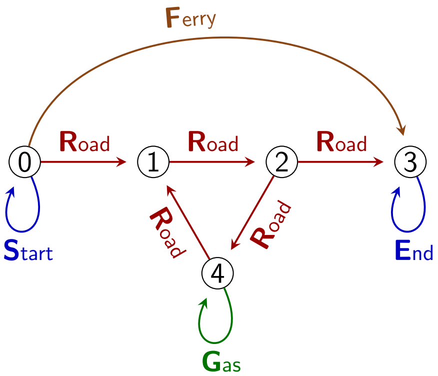
Matching $Q_2 = \mathbf{G}$

$L(Q_2) = \{\mathbf{G}\}$

Match for Q_2	Label
$4 \to 4$	**G**

$Q_3 = \mathbf{R} + \mathbf{F}$

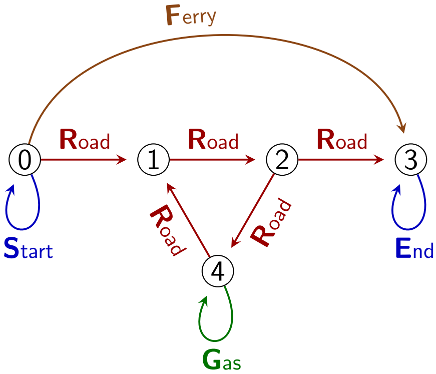
$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$



$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

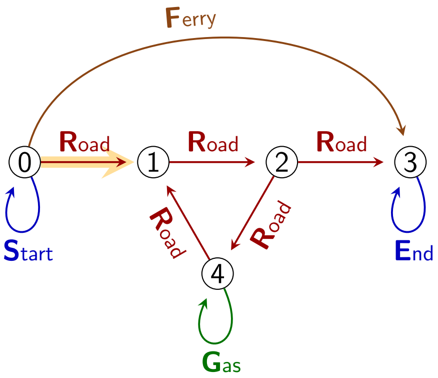


$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



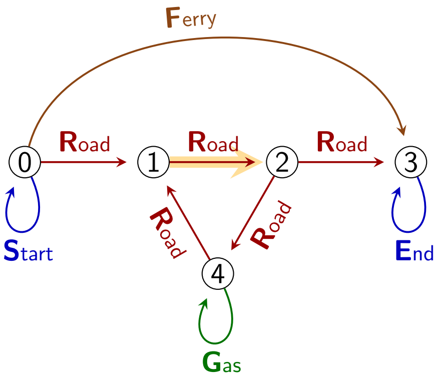
Disjunction

$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



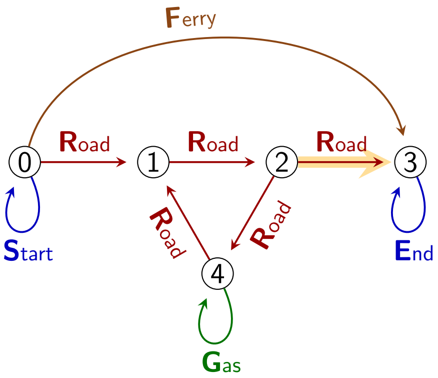
Disjunction

$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**

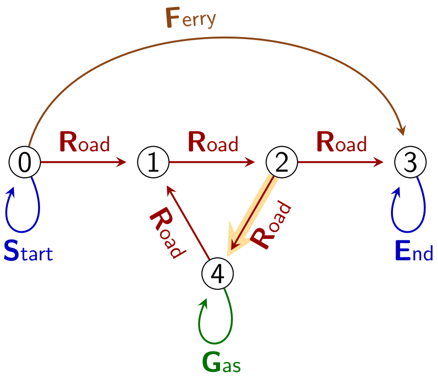


$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



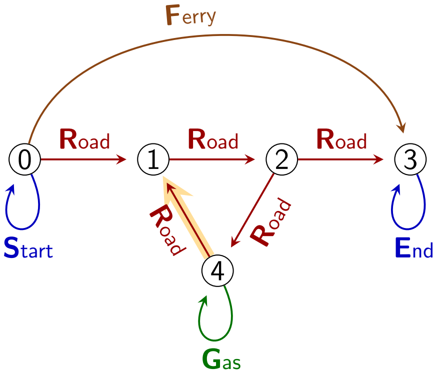
Disjunction

$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**

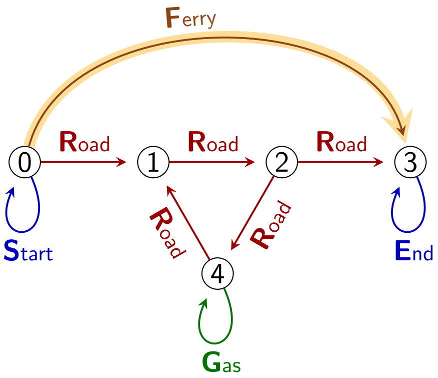


$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**

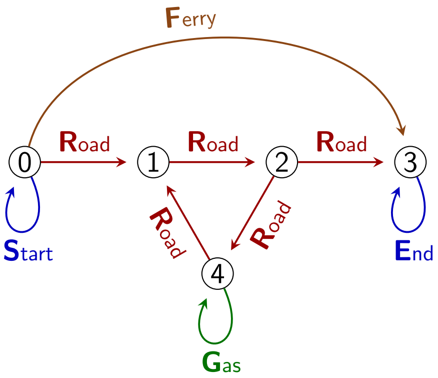


$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

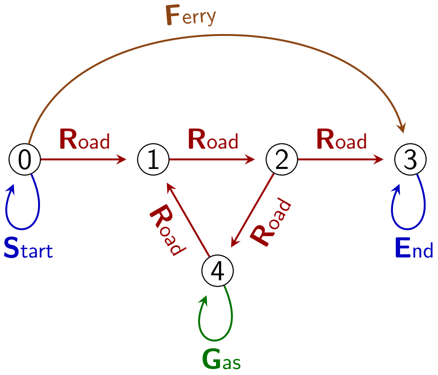
Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

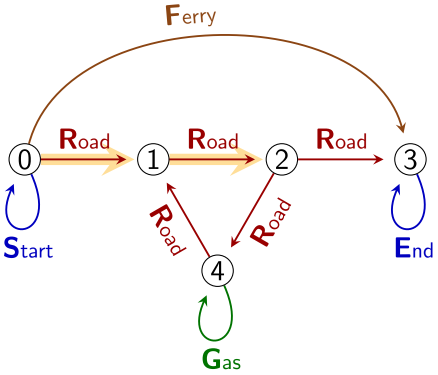


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \rightarrow 1 \rightarrow 2$	**RR**

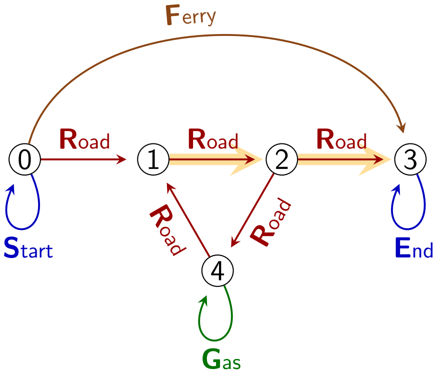


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**

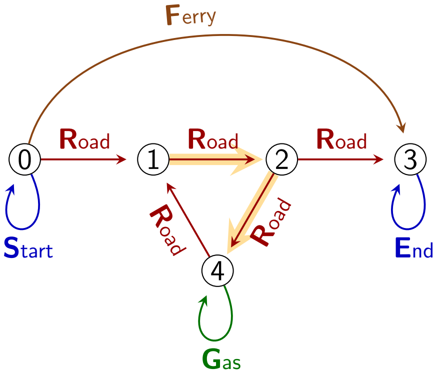


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**

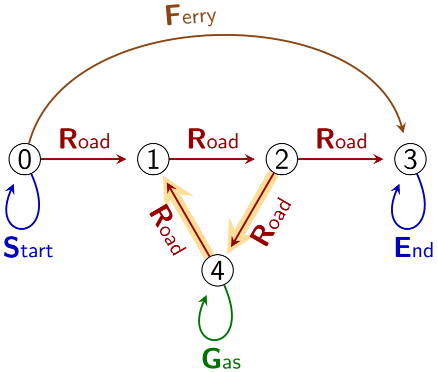


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**

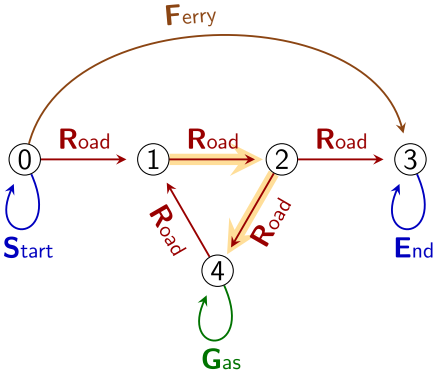


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**
$4 \to 1 \to 2$	**RR**



Concatenation

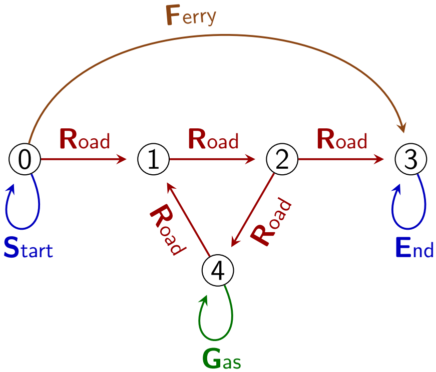
$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**
$4 \to 1 \to 2$	**RR**

Matches for $Q_5 = \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{R} \cdot \mathbf{R}$

$L(Q_5) = \{\mathbf{SRRR}\}$



Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

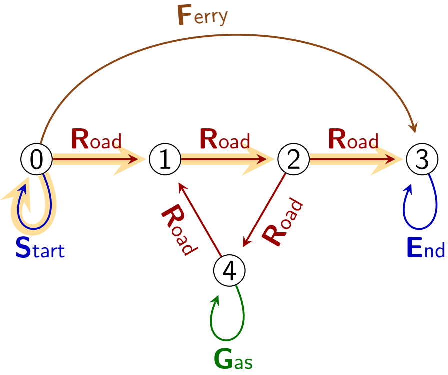
$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**
$4 \to 1 \to 2$	**RR**

Matches for $Q_5 = \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{R} \cdot \mathbf{R}$

$L(Q_5) = \{\mathbf{SRRR}\}$

$0 \to 0 \to 1 \to 2 \to 3$	**SRRR**
$0 \to 0 \to 1 \to 2 \to 3$	**SRRR**



Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

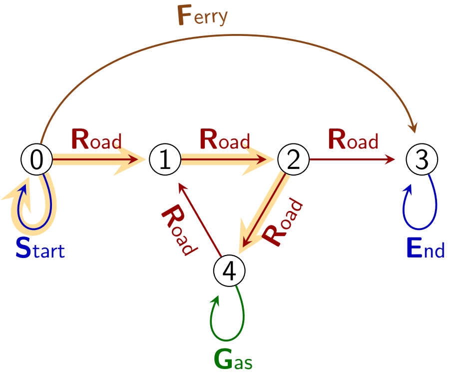
$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**
$4 \to 1 \to 2$	**RR**

Matches for $Q_5 = \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{R} \cdot \mathbf{R}$

$L(Q_5) = \{\mathbf{SRRR}\}$

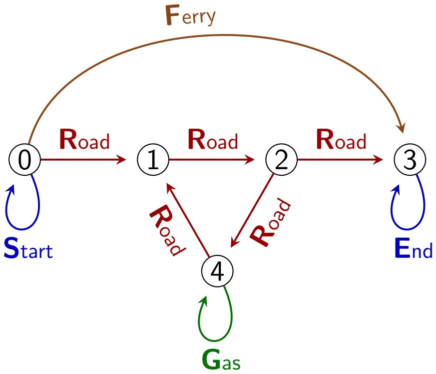
$0 \to 0 \to 1 \to 2 \to 3$	**SRRR**
$0 \to 0 \to 1 \to 2 \to 3$	**SRRR**



Concatenation & disjunction

$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

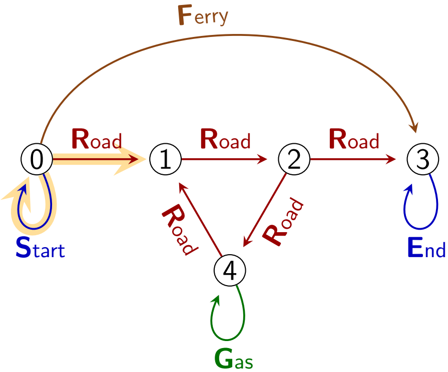
$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$



$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

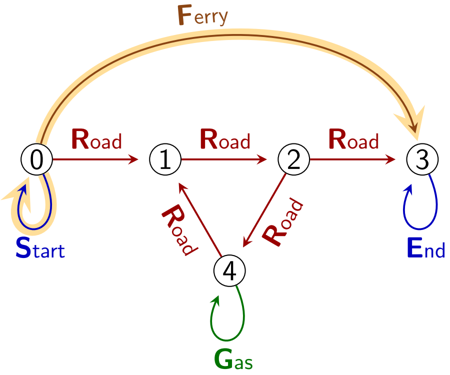
Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**



$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**



Concatenation & disjunction

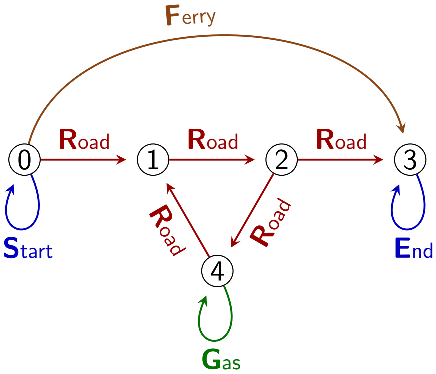
$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

$Q_7 = (\mathbf{S} + \mathbf{R})(\mathbf{F} + \mathbf{G})(\mathbf{E} + \mathbf{R})$

$L(Q_7) =$



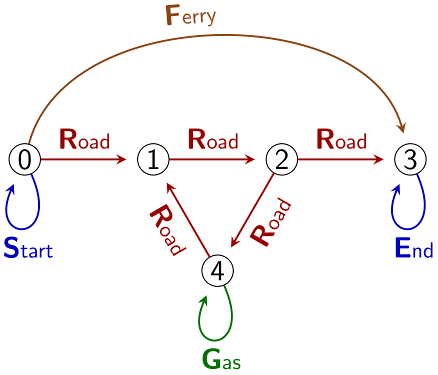
$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

$Q_7 = (\mathbf{S} + \mathbf{R})(\mathbf{F} + \mathbf{G})(\mathbf{E} + \mathbf{R})$

$L(Q_7) = \{\mathbf{SFE}, \mathbf{SFR}, \mathbf{SGE}, \mathbf{SGR}, \mathbf{RFE}, \mathbf{RFR}, \mathbf{RGE}, \mathbf{RGR}\}$



Concatenation & disjunction

$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

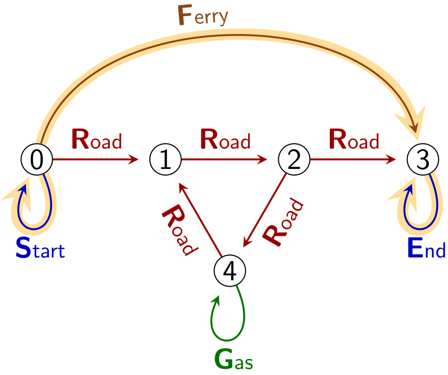
$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

$Q_7 = (\mathbf{S} + \mathbf{R})(\mathbf{F} + \mathbf{G})(\mathbf{E} + \mathbf{R})$

$L(Q_7) = \{\mathbf{SFE}, \mathbf{SFR}, \mathbf{SGE}, \mathbf{SGR}, \mathbf{RFE}, \mathbf{RFR}, \mathbf{RGE}, \mathbf{RGR}\}$

Match for Q_7	Label
$0 \to 0 \to 3 \to 3$	**SFE**
$2 \to 4 \to 4 \to 1$	**RGR**



Concatenation & disjunction

$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

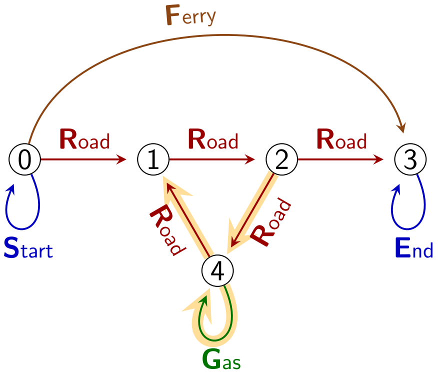
$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

$Q_7 = (\mathbf{S} + \mathbf{R})(\mathbf{F} + \mathbf{G})(\mathbf{E} + \mathbf{R})$

$L(Q_7) = \{\mathbf{SFE}, \mathbf{SFR}, \mathbf{SGE}, \mathbf{SGR}, \mathbf{RFE}, \mathbf{RFR}, \mathbf{RGE}, \mathbf{RGR}\}$

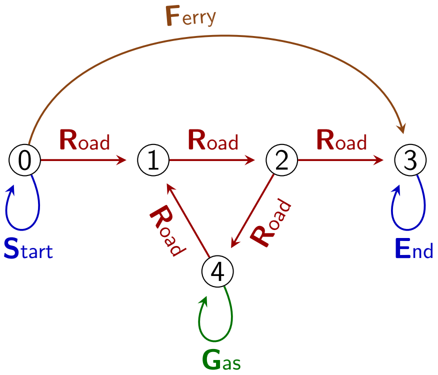
Match for Q_7	Label
$0 \to 0 \to 3 \to 3$	**SFE**
$2 \to 4 \to 4 \to 1$	**RGR**



Kleene star

$Q_8 = \mathbf{R}^*$

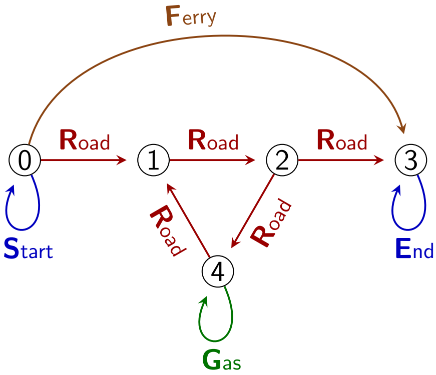
$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$



Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$



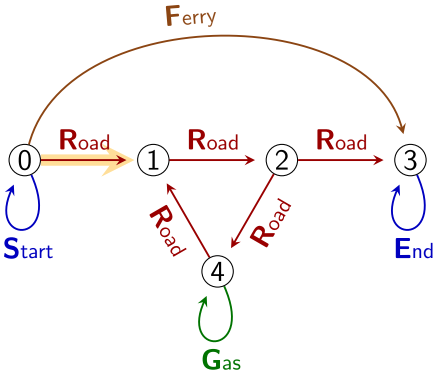
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
⋮	
$2 \to 4 \to 1$	**RR**
⋮	
$1 \to 2 \to 4 \to 1$	**RRR**
⋮	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
⋮	

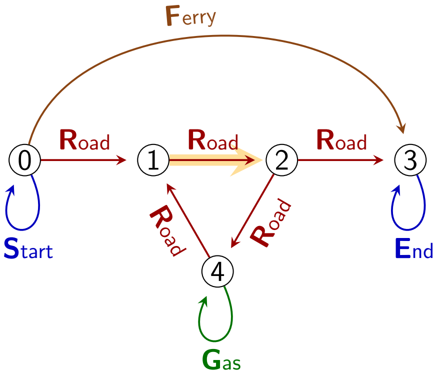


⚠ $L(Q_8)$ is infinite ⚠

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
⋮	
$2 \to 4 \to 1$	**RR**
⋮	
$1 \to 2 \to 4 \to 1$	**RRR**
⋮	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
⋮	



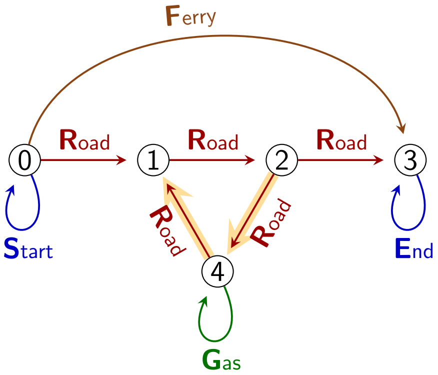
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
⋮	
$2 \to 4 \to 1$	**RR**
⋮	
$1 \to 2 \to 4 \to 1$	**RRR**
⋮	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
⋮	

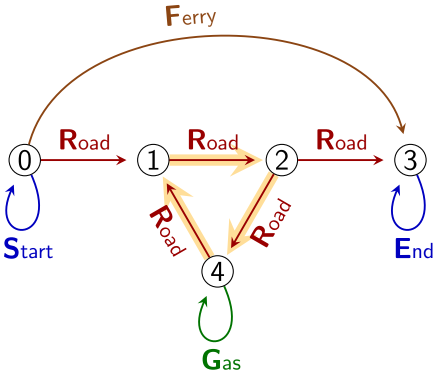


⚠ $L(Q_8)$ is infinite ⚠

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
$\vdots$	

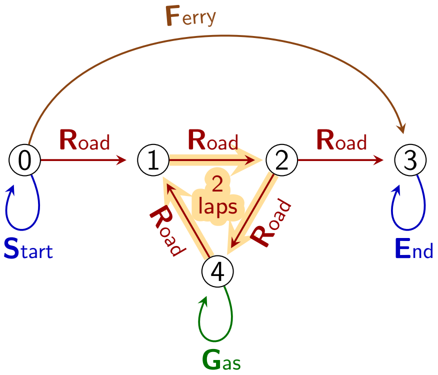


⚠ $L(Q_8)$ is infinite ⚠

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
⋮	
$2 \to 4 \to 1$	**RR**
⋮	
$1 \to 2 \to 4 \to 1$	**RRR**
⋮	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
⋮	



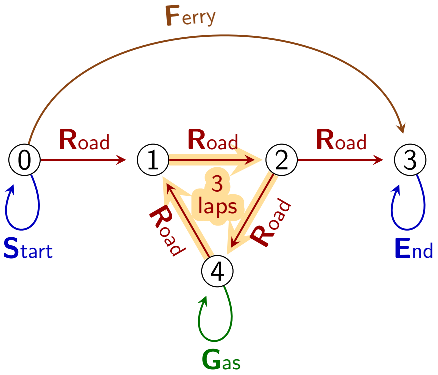
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \dots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
⋮	
$2 \to 4 \to 1$	**RR**
⋮	
$1 \to 2 \to 4 \to 1$	**RRR**
⋮	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
⋮	

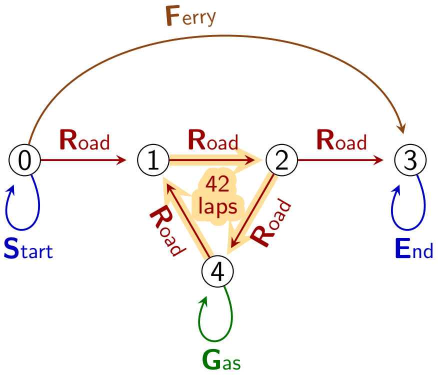


⚠ $L(Q_8)$ is infinite ⚠

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
⋮	
$2 \to 4 \to 1$	**RR**
⋮	
$1 \to 2 \to 4 \to 1$	**RRR**
⋮	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
⋮	

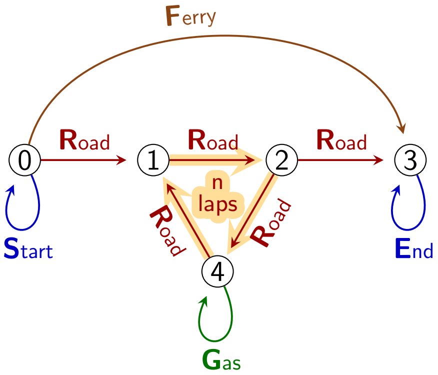


⚠ $L(Q_8)$ is infinite ⚠

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
⋮	
$2 \to 4 \to 1$	**RR**
⋮	
$1 \to 2 \to 4 \to 1$	**RRR**
⋮	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
⋮	



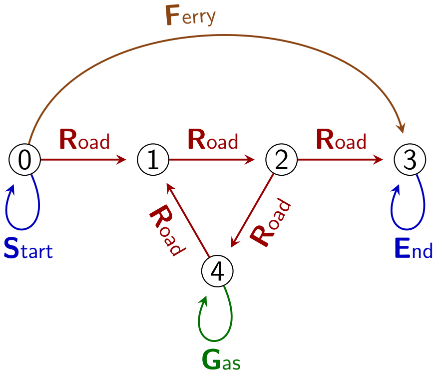
⚠ $L(Q_8)$ is infinite ⚠

⚠ Infinitely many matches ⚠

Computing matches

Exercice

Compute the matches to query $Q_9 = (\mathbf{R} + \mathbf{F})^*\mathbf{G}(\mathbf{R} + \mathbf{F})^*$ that start in 0 and end in 3.



Computing matches

Exercice

Compute the matches to query $Q_9 = (\mathbf{R} + \mathbf{F})^*\mathbf{G}(\mathbf{R} + \mathbf{F})^*$ that start in 0 and end in 3.

Answer

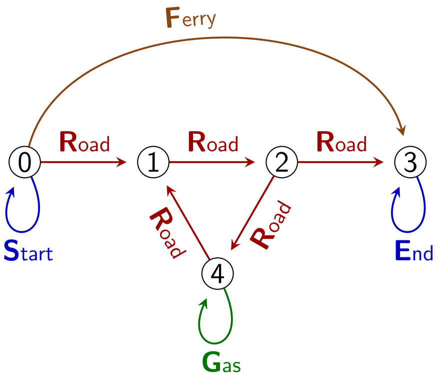
$$0 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 4$$

$$\left(\xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 4\right)^*$$

$$\xrightarrow{\mathbf{G}} 4$$

$$\left(\xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 4\right)^*$$

$$\xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 3$$



Any idea an how to compute matches in general?

Regexps may be transformed into a finite automaton

Glushkov Construction*

Input a regexp Q

Output a nondeterministic automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L(Q)$

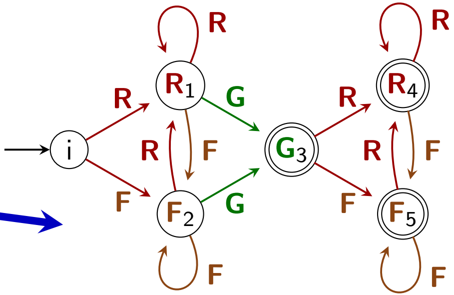
Properties of $\mathcal{A}$

- $\mathcal{D}$ is small: the number of state is in $O(\text{size}(Q))$
- $\mathcal{D}$ is computed efficiently $O(\text{size}(Q)^2)$
- $\mathcal{D}$ has no epsilon-transitions

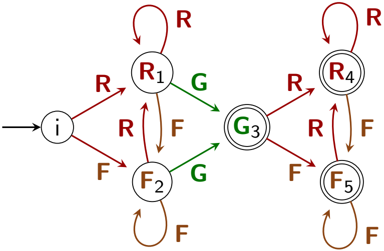
*Other names: position automaton, standard automaton, Berry-Sethi construction

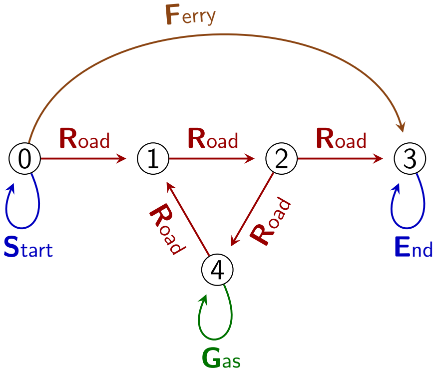
$(\mathbf{R} + \mathbf{F})^*\mathbf{G}(\mathbf{R} + \mathbf{F})^*$

$(\mathbf{R}_1 + \mathbf{F}_2)^*\mathbf{G}_3(\mathbf{R}_4 + \mathbf{F}_5)^*$



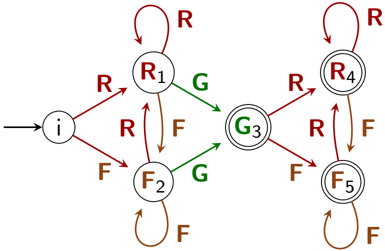
A graph is essentially an automaton

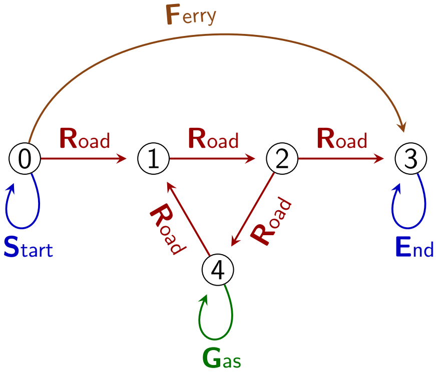




A graph is essentially an automaton

Exercice: compute the product graph×query

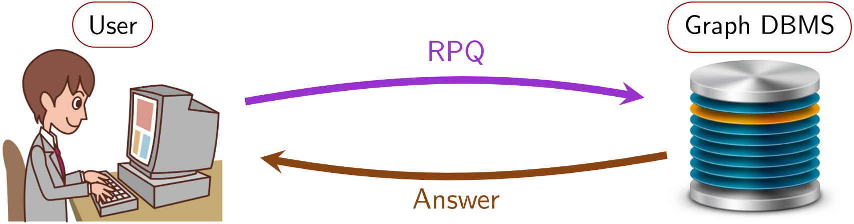




Part I: Theoretical foundations

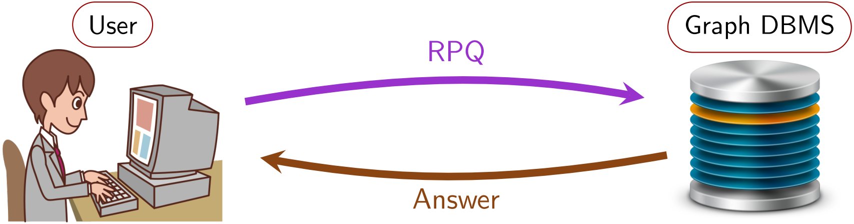
4. The most common RPQ semantics

Computing a finite answer



Infinitely many matches but the user expects **finite** answer

Computing a finite answer



Infinitely many matches but the user expects **finite** answer

Different semantics for RPQs

- A **RPQ semantics** = a way to interpret RPQs
- The semantics defines the **correct answer**
  $\Rightarrow$ The same query has different answers under different semantics
- Goal of an RPQ semantics: ensure the answer to be **finite**, while remaining **meaningful** and **easy to compute**.

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1,3),(2,2),(0,3)\}$

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1,3),(2,2),(0,3)\}$

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 3$	(1,3)
$2 \rightarrow 2$	(2,2)
$0 \rightarrow 0 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow 3$	(0,3)
$1 \rightarrow 0 \rightarrow 3$	(1,3)

Full answer is: $\{(1,3),(2,2),(0,3)\}$

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1,3),(2,2),(0,3)\}$

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1, 3), (2, 2), (0, 3)\}$

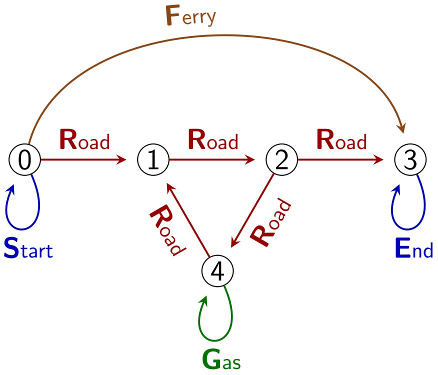
Endpoint semantics (2)

Evaluating a reachability query

$Q_{10} = \mathbf{GR}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$ $\to 4 \to 1 \to 2$ $\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer

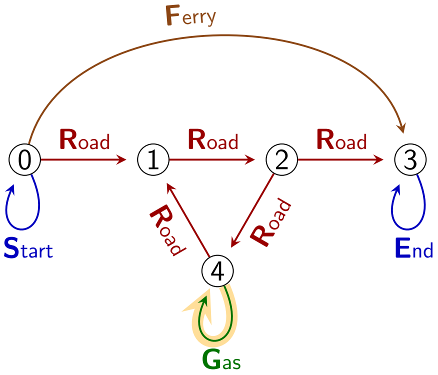


Answer to Q_{10} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

$Q_{10} = \mathbf{GR}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
$\vdots$	$\vdots$
$4 \to 4 \to 1 \to 2 \to 4 \to 1 \to 2 \to 3$	(4,3)
$\vdots$	$\vdots$

Other matches do not add new pairs to the answer



Answer to Q_{10} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

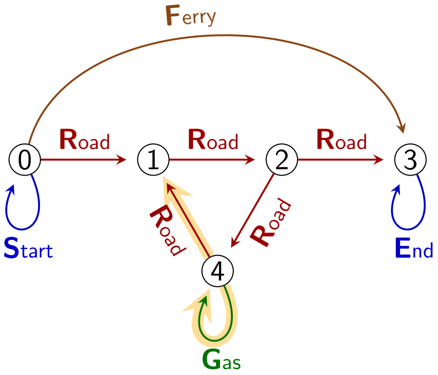
Endpoint semantics (2)

Evaluating a reachability query

$Q_{10} = \mathbf{GR}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{10} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

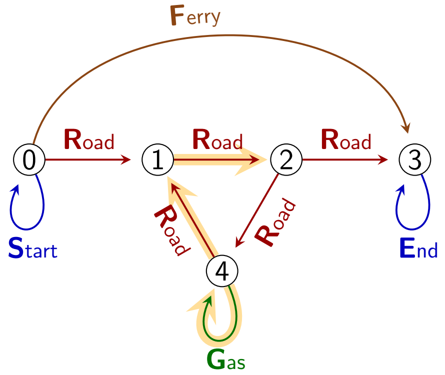
Endpoint semantics (2)

Evaluating a reachability query

$Q_{10} = \mathbf{GR}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$ $\to 4 \to 1 \to 2$ $\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{10} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

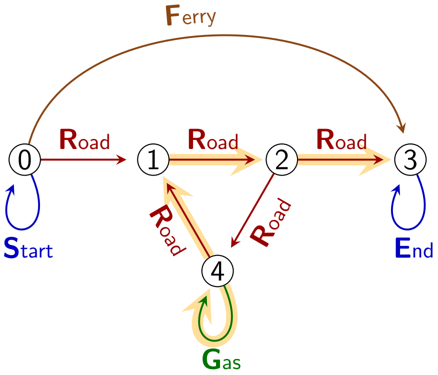
Endpoint semantics (2)

Evaluating a reachability query

$Q_{10} = \mathbf{G}\mathbf{R}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$ $\to 4 \to 1 \to 2$ $\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{10} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

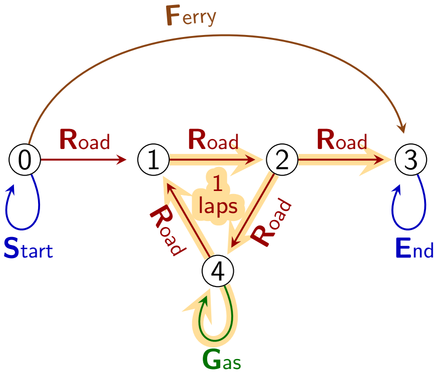
Endpoint semantics (2)

Evaluating a reachability query

$Q_{10} = \mathbf{G}\mathbf{R}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$	
$\to 4 \to 1 \to 2$	
$\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{10} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

Endpoint semantics (3)

Pros and cons

Pros

- Efficient algorithms
- Output is always small
- Well grounded theory

Endpoint semantics (3)

Pros and cons

Pros

- Efficient algorithms
- Output is always small
- Well grounded theory

Cons

- Very limited information in the answer
  - User: *"I want to go from Paris to Lyon by car"*
  - Database: *"Yes you can"*

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = smallest number of edges

Example

Match	Endpoints	Length	
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3	Shortest for $(1, 3)$
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4	Not shortest for $(1, 3)$
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3	Not shortest for $(0, 3)$
$0 \to 2 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$
$0 \to 0 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$

Full answer: $\{1 \to 0 \to 2 \to 3, \quad 0 \to 2 \to 3, \quad 0 \to 0 \to 3\}$

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = smallest number of edges

Example

Match	Endpoints	Length	
$1 \to 0 \to 2 \to 3$	$(1,3)$	3	Shortest for $(1,3)$
$1 \to 0 \to 2 \to 2 \to 3$	$(1,3)$	4	Not shortest for $(1,3)$
$0 \to 2 \to 2 \to 3$	$(0,3)$	3	Not shortest for $(0,3)$
$0 \to 2 \to 3$	$(0,3)$	2	Tied shortest for $(0,3)$
$0 \to 0 \to 3$	$(0,3)$	2	Tied shortest for $(0,3)$

Full answer: $\{1 \to 0 \to 2 \to 3, \quad 0 \to 2 \to 3, \quad 0 \to 0 \to 3\}$

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = smallest number of edges

Example

Match	Endpoints	Length	
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3	Shortest for $(1, 3)$
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4	Not shortest for $(1, 3)$
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3	Not shortest for $(0, 3)$
$0 \to 2 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$
$0 \to 0 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$

Full answer: $\{1 \to 0 \to 2 \to 3, \quad 0 \to 2 \to 3, \quad 0 \to 0 \to 3\}$

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = smallest number of edges

Example

Match	Endpoints	Length	
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3	Shortest for $(1, 3)$
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4	Not shortest for $(1, 3)$
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3	Not shortest for $(0, 3)$
$0 \to 2 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$
$0 \to 0 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$

Full answer: $\{1 \to 0 \to 2 \to 3, \quad 0 \to 2 \to 3, \quad 0 \to 0 \to 3\}$

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = smallest number of edges

Example

Match	Endpoints	Length	
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3	Shortest for $(1, 3)$
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4	Not shortest for $(1, 3)$
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3	Not shortest for $(0, 3)$
$0 \to 2 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$
$0 \to 0 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$

Full answer: $\{1 \to 0 \to 2 \to 3,\ \ 0 \to 2 \to 3,\ \ 0 \to 0 \to 3\}$

Shortest semantics (2)

Evaluating a reachability query

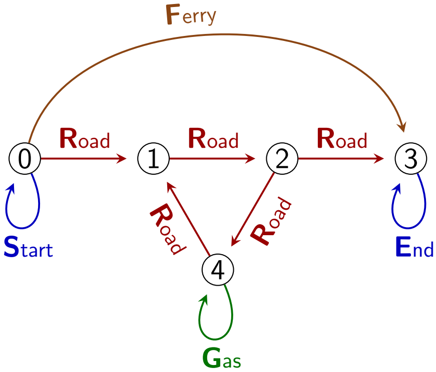
$Q_{11} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)

Example of discarded match

$4 \to 4 \to 1 \to 2 \to 4$ is not in the answer because it is longer than $4 \to 4$



Shortest semantics (2)

Evaluating a reachability query

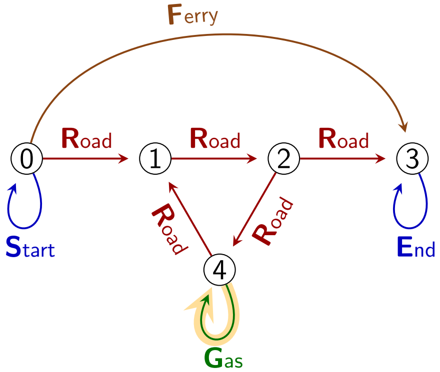
$Q_{11} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
4 → 4	(4,4)
4 → 4 → 1	(4,1)
4 → 4 → 1 → 2	(4,2)
4 → 4 → 1 → 2 → 3	(4,3)

Example of discarded match

4 → 4 → 1 → 2 → 4 is not in the answer because it is longer than 4 → 4



Shortest semantics (2)

Evaluating a reachability query

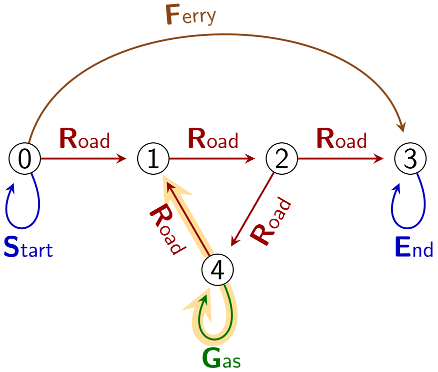
$Q_{11} = \mathbf{G}\mathbf{R}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)

Example of discarded match

$4 \to 4 \to 1 \to 2 \to 4$ is not in the answer because it is longer than $4 \to 4$



Shortest semantics (2)

Evaluating a reachability query

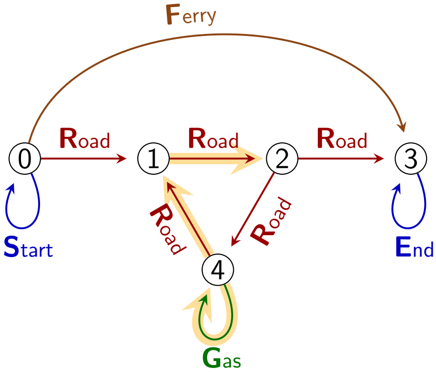
$Q_{11} = \mathbf{G}\mathbf{R}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)

Example of discarded match

$4 \to 4 \to 1 \to 2 \to 4$ is not in the answer because it is longer than $4 \to 4$



Shortest semantics (2)

Evaluating a reachability query

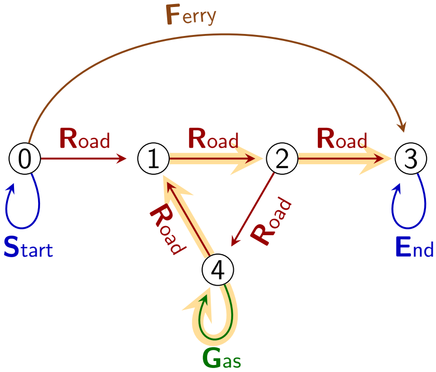
$Q_{11} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)

Example of discarded match

$4 \to 4 \to 1 \to 2 \to 4$ is not in the answer because it is longer than $4 \to 4$



Shortest semantics (2)

Evaluating a reachability query

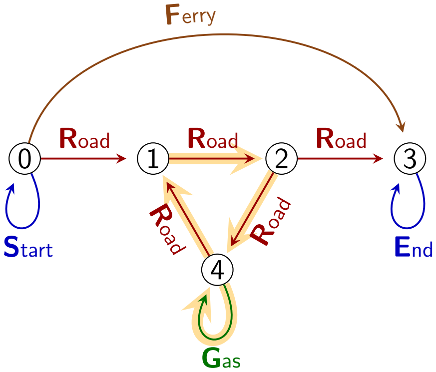
$Q_{11} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)

Example of discarded match

$4 \to 4 \to 1 \to 2 \to 4$ is not in the answer because it is longer than $4 \to 4$



Shortest semantics (3)

Exercice: evaluating some queries

$Q_{12} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

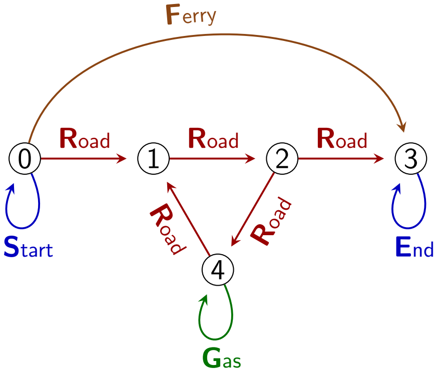
Answer to Q_{12}:

?

$Q_{13} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{13}:

?



Shortest semantics (3)

Exercice: evaluating some queries

$Q_{12} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

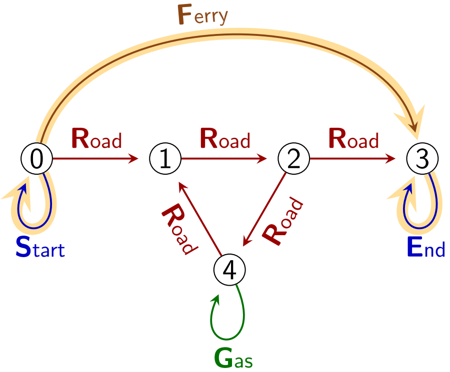
Answer to Q_{12}:

$\{ 0 \rightarrow 0 \rightarrow 3 \rightarrow 3 \}$

$Q_{13} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{13}:

?



Shortest semantics (3)

Exercice: evaluating some queries

$Q_{12} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

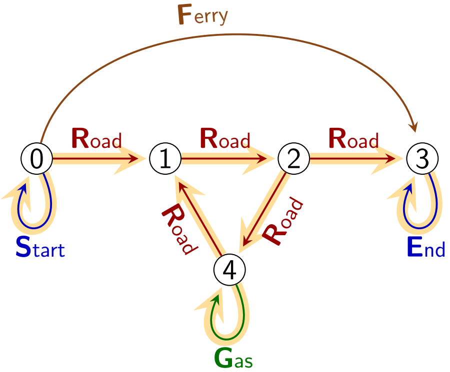
Answer to Q_{12}:

$$\{\ 0 \rightarrow 0 \rightarrow 3 \rightarrow 3\ \}$$

$Q_{13} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{13}:

$$\{\ 0 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 3\ \}$$



Shortest semantics (4)

Pros and con

Pros

- Returns walks
- Efficient algorithms (BFS in the product graph$\times$query)
- If there are matches from s to t, at least one of them is in the answer

Shortest semantics (4)

Pros and con

Pros

- Returns walks
- Efficient algorithms (BFS in the product graph$\times$query)
- If there are matches from s to t, at least one of them is in the answer

Cons

- The shortest walk is not always the "best"
  - *"Do we always want to take the ferry over the direct road?"*
  - (Real query languages allow to assign costs to edges/atoms)
- No vertical post-processing
  - Vertical $=$ accross the walks with the same endpoints
  - *"What is the average time?"*
  - *"What is the connectedness level?"*

Trail semantics (1)

Used by Cypher (Neo4j) and GQL with keyword `ALL TRAIL`

Trail semantics (1)

Used by Cypher (Neo4j) and GQL with keyword `ALL TRAIL`

Principle

- Return a set of walks
- Apply a filter on the set of matching walks
- The filter is: each walk that repeats an edge is filtered out

Examples

Match	Decision
$1 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 3$	No repetition $\Rightarrow$ Kept in the answer
$1 \rightarrow 0 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow 2$	Repeated edges $\Rightarrow$ Filtered out

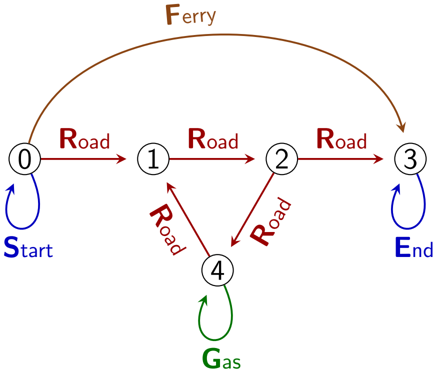
Trail semantics (2)

Evaluating Q_{14}

$$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$

Applying the filter

Matches	Keep?
The ferry walk	
The straight road	
The road with 1 lap	
The road with 2 laps	
⋮	



Answer of Q_1 under trail semantics:

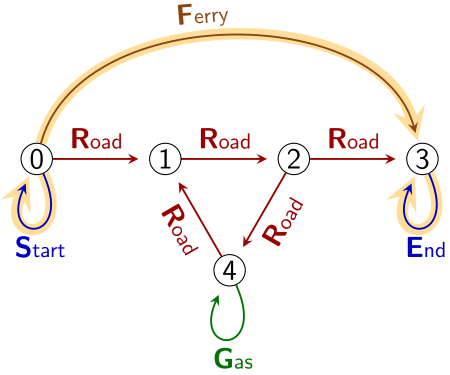
$\{ \quad \}$

Evaluating Q_{14}

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Applying the filter

Matches	Keep?
The ferry walk	
The straight road	
The road with 1 lap	
The road with 2 laps	
⋮	



Answer of Q_1 under trail semantics:

{ }

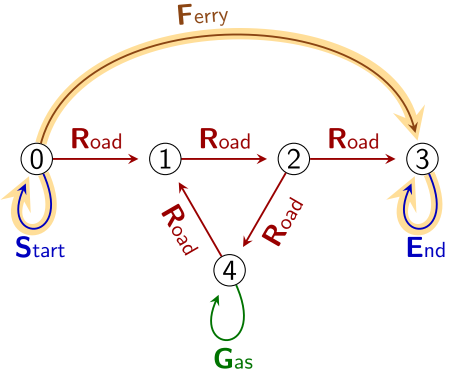
Trail semantics (2)

Evaluating Q_{14}

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	
The road with 1 lap	
The road with 2 laps	
⋮	



Answer of Q_1 under trail semantics:

{ $0 \to 0 \to 3 \to 3$ }

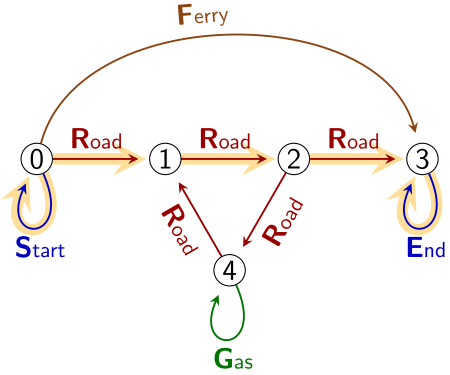
Trail semantics (2)

Evaluating Q_{14}

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	
The road with 1 lap	
The road with 2 laps	
⋮	



Answer of Q_1 under trail semantics:

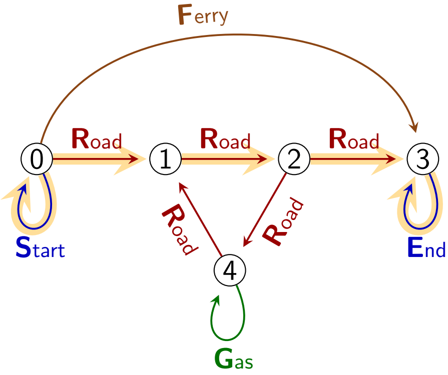
$\{ \quad 0 \to 0 \to 3 \to 3 \quad \}$

Evaluating Q_{14}

$$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	
The road with 2 laps	
⋮	



Answer of Q_1 under trail semantics:

$$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$$

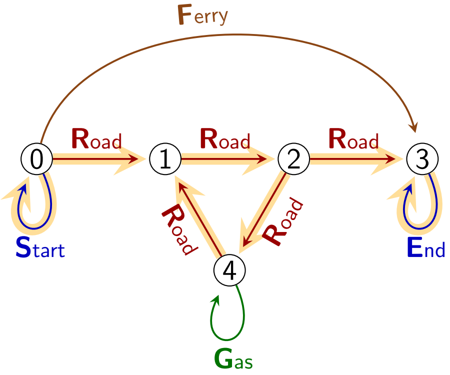
Trail semantics (2)

Evaluating Q_{14}

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Applying the filter

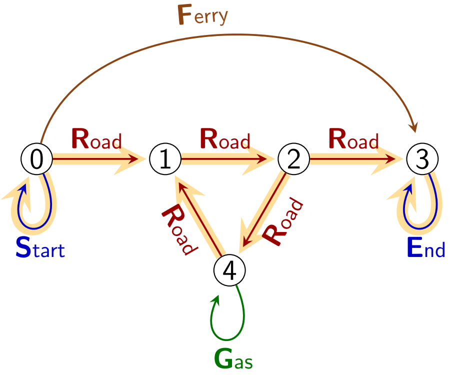
Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	
The road with 2 laps	
⋮	



Answer of Q_1 under trail semantics:

$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$



Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	No
The road with 2 laps	
⋮	

Answer of Q_1 under trail semantics:

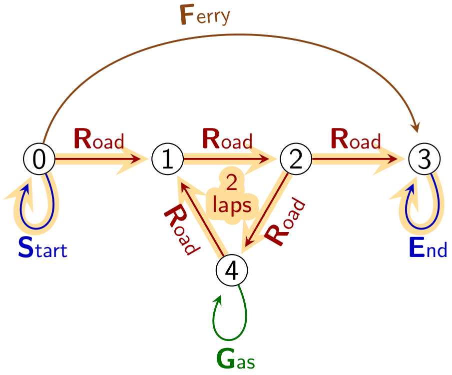
$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$

Evaluating Q_{14}

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	No
The road with 2 laps	
⋮	



Answer of Q_1 under trail semantics:

$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$

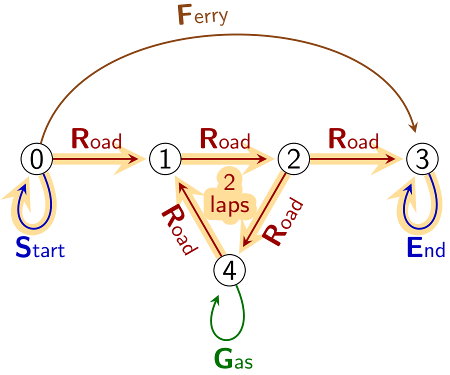
Trail semantics (2)

Evaluating Q_{14}

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	No
The road with 2 laps	No
⋮	



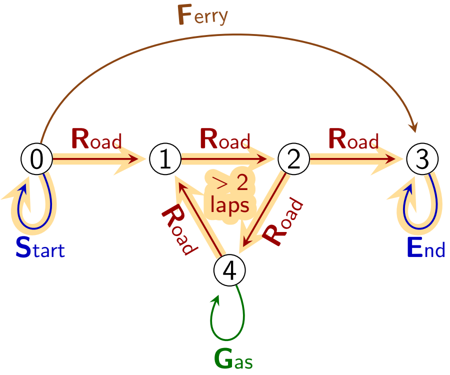
Answer of Q_1 under trail semantics:

$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	No
The road with 2 laps	No
⋮	No



Answer of Q_1 under trail semantics:

$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$

Exercice: evaluating some queries

$Q_{15} = \mathbf{G}\mathbf{R}^*$

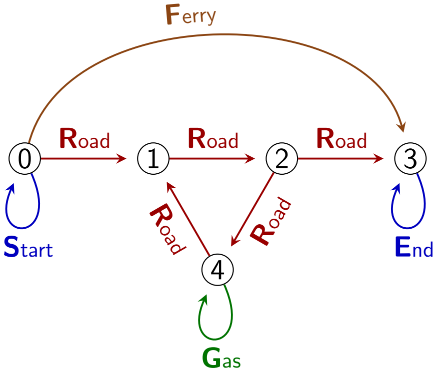
Answer to Q_{15}:

?

$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

?



Exercice: evaluating some queries

$Q_{15} = \mathbf{G}\mathbf{R}^*$

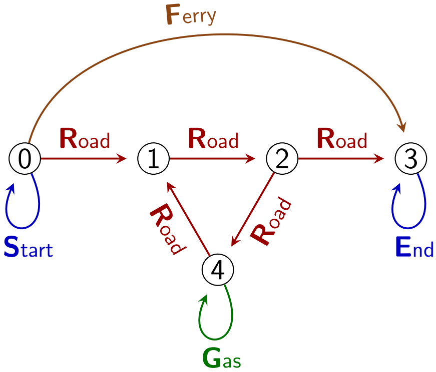
Answer to Q_{15}:

$$\{ \quad 4 \rightarrow 4\ , \quad 4 \rightarrow 4 \rightarrow 1\ , \quad 4 \rightarrow 4 \rightarrow 1 \rightarrow 2\ , \quad 4 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 3\ , \quad 4 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 4 \quad \}$$

$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

?

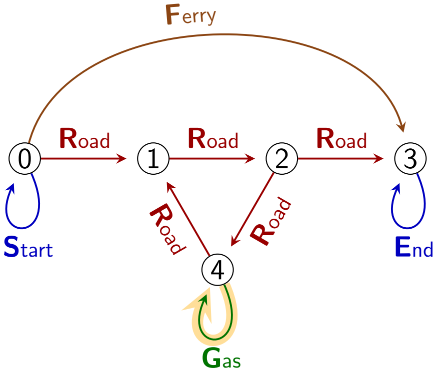


Exercice: evaluating some queries

$Q_{15} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{15}:

$$\{\ 4 \to 4,\ 4 \to 4 \to 1,\ 4 \to 4 \to 1 \to 2,\ 4 \to 4 \to 1 \to 2 \to 3,\ 4 \to 4 \to 1 \to 2 \to 4\ \}$$



$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

?

Exercice: evaluating some queries

$Q_{15} = \mathbf{GR}^*$

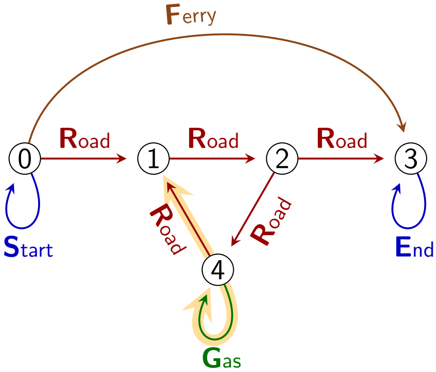
Answer to Q_{15}:

$$\{ \; 4 \to 4 \, , \; 4 \to 4 \to 1 \, , \; 4 \to 4 \to 1 \to 2 \, , \; 4 \to 4 \to 1 \to 2 \to 3 \, , \; 4 \to 4 \to 1 \to 2 \to 4 \; \}$$

$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

?



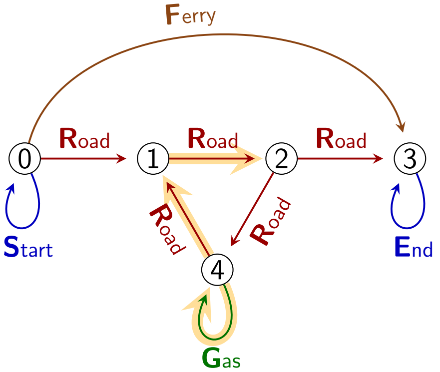
Trail semantics (3)

Exercice: evaluating some queries

$Q_{15} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{15}:

$$\{ \; 4 \to 4 \; , \; 4 \to 4 \to 1 \; , \; 4 \to 4 \to 1 \to 2 \; , \; 4 \to 4 \to 1 \to 2 \to 3 \; , \; 4 \to 4 \to 1 \to 2 \to 4 \; \}$$



$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

?

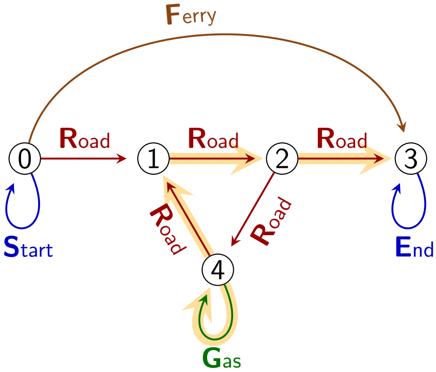
Trail semantics (3)

Exercice: evaluating some queries

$Q_{15} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{15}:

$$\{\ 4 \to 4\ ,\quad 4 \to 4 \to 1\ ,\quad 4 \to 4 \to 1 \to 2\ ,\quad 4 \to 4 \to 1 \to 2 \to 3\ ,\quad 4 \to 4 \to 1 \to 2 \to 4\ \}$$



$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

?

Exercice: evaluating some queries

$Q_{15} = \mathbf{G}\mathbf{R}^*$

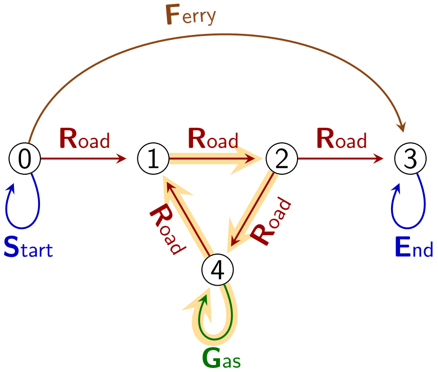
Answer to Q_{15}:

$$\{\ 4 \to 4\ ,\ 4 \to 4 \to 1\ ,\ 4 \to 4 \to 1 \to 2\ ,\ 4 \to 4 \to 1 \to 2 \to 3\ ,\ 4 \to 4 \to 1 \to 2 \to 4\ \}$$

$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

?



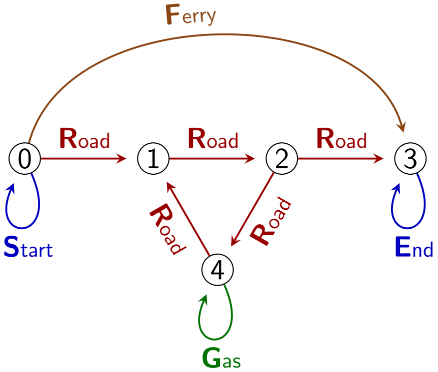
Trail semantics (3)

Exercice: evaluating some queries

$Q_{15} = \mathbf{GR}^*$

Answer to Q_{15}:

$$\{ \quad 4 \to 4\ , \quad 4 \to 4 \to 1\ , \quad 4 \to 4 \to 1 \to 2\ , \quad 4 \to 4 \to 1 \to 2 \to 3\ , \quad 4 \to 4 \to 1 \to 2 \to 4 \quad \}$$



$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

?

Trail semantics (3)

Exercice: evaluating some queries

$Q_{15} = \mathbf{GR}^*$

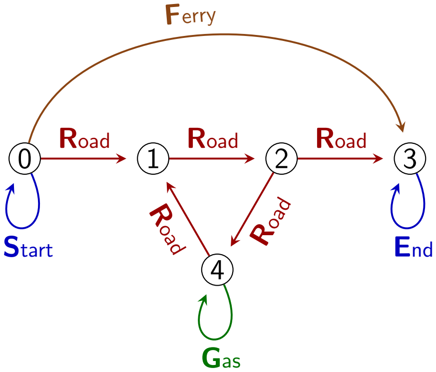
Answer to Q_{15}:

$$\{\ 4 \to 4\ ,\ 4 \to 4 \to 1\ ,\ 4 \to 4 \to 1 \to 2\ ,\ 4 \to 4 \to 1 \to 2 \to 3\ ,\ 4 \to 4 \to 1 \to 2 \to 4\ \}$$

$Q_{16} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{16}:

$\emptyset$



Trail semantics (4)

Pros and cons

Pros

- Returns walks
- Easy to explain
- Enable vertical post-processing
  - Vertical = accross the walks with the same endpoints
  - *"What is the average time?"*
  - *"What is the connectedness level?"*

Trail semantics (4)

Pros and cons

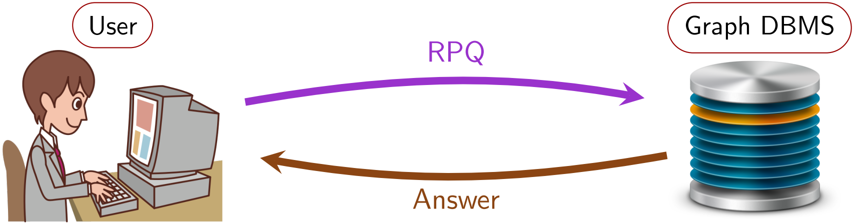
Pros

- Returns walks
- Easy to explain
- Enable vertical post-processing
  - Vertical = accross the walks with the same endpoints
  - *"What is the average time?"*
  - *"What is the connectedness level?"*

Cons

- **Inefficient** in bad cases.
  Ex: checking whether $\mathbf{R}^*\mathbf{G}\mathbf{R}^*$ returns anything is NP-hard
- "No repeated edge" is a filter that is sometimes **counterintuitive**
  Ex: $\mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$ had matches but the answer is empty

Computing a finite answer

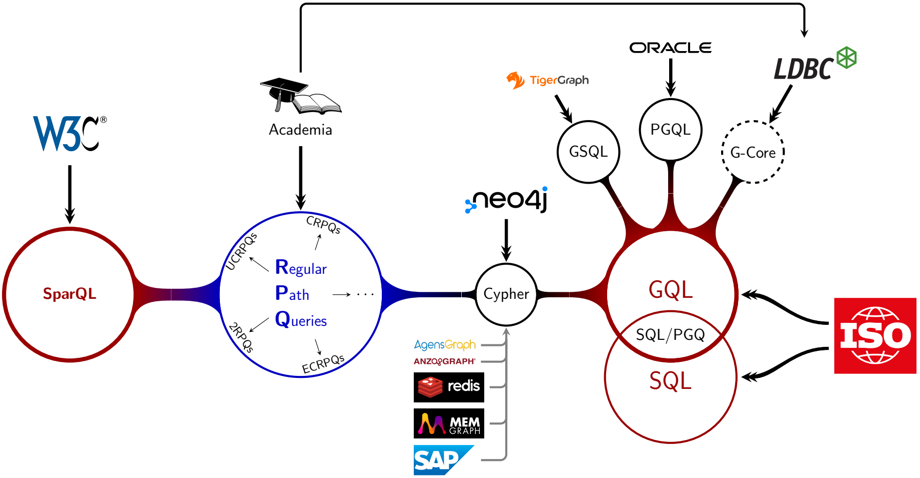


Infinitely many matches but the user expects **finite** answer

Different semantics for RPQs

- Endpoint → Filters out all navigational information
- Shortest → No vertical postprocessing and arbitrary metrics
- Trail → Inefficient and sometimes discard meaningful matches

⟹ No RPQ semantics is clearly superior



- SparQL and most academic work on RPQs use endpoint semantics
- Cypher uses trail semantics
- GSQL, PGQL and G-Core uses shortest semantics (and variants)
- GQL and SQL/PGQ allow to switch between many RPQ semantics

Part II: Neo4j, Property graphs and Cypher

Part **II**: Neo4j, Property graphs and Cypher

1. Data model: Property graphs

Components of a property graph

A **node** (≈**vertex**) encodes a complex values.
It bears **labels** are for grouping.

Ex: t carries `Teacher`, `Person`
c carries `Course`

A **Relation** (≈**edge**) connects **nodes**
It bears one **type** (≈**label**) provides the nature of the relation

Ex: $e = t \xrightarrow{\texttt{TEACHES}} c$

A **property** describes an aspect of a **node** or an **relation**
It maps

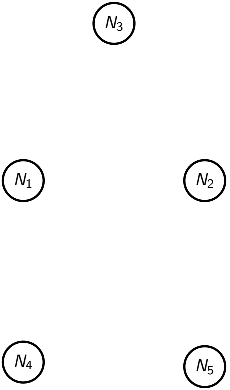
- a **key** (described aspect)
- to a **pure value** (description)

Ex: t has `name:"Victor"`
e has `since:`2023

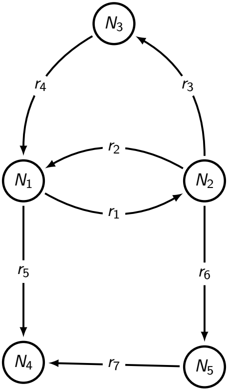
A **pure value** (int, string, etc) contains all the information about itself.

Ex: `"Victor"` has 6 letters

First example of a property graph

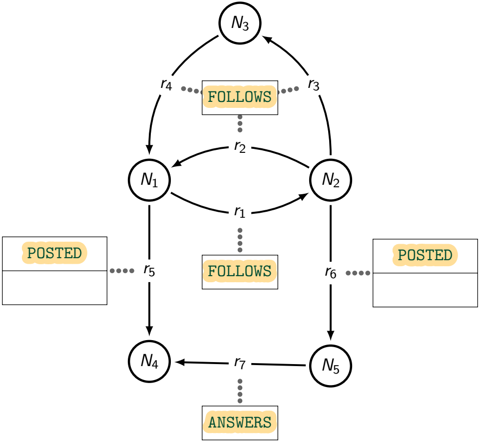


- Nodes : $N_1, N_2, \cdots, N_5$



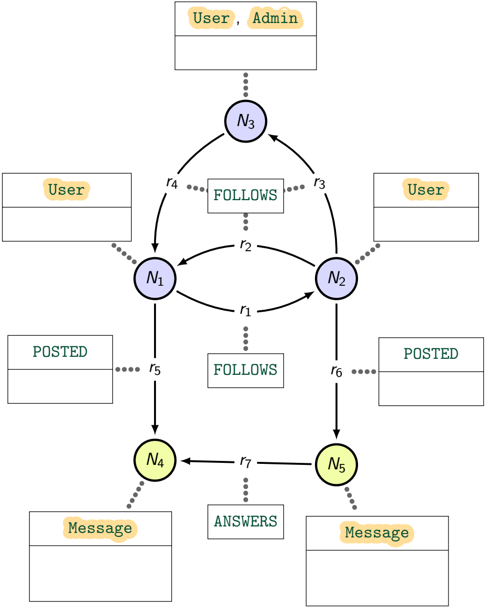
- Nodes : $N_1, N_2, \cdots, N_5$
- Relations : $r_1, r_2, \cdots, r_7$

First example of a property graph



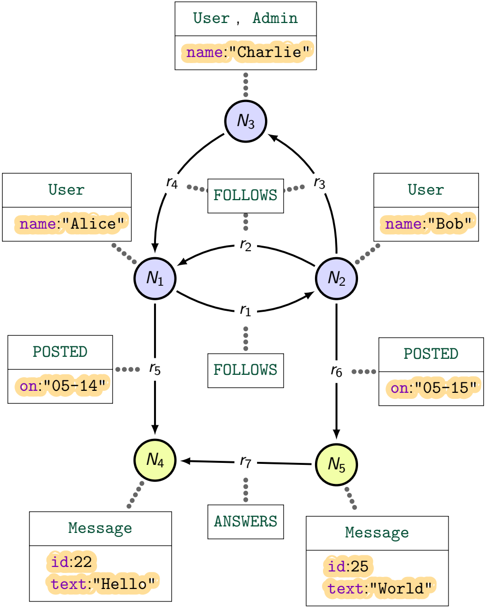
- Nodes : $N_1, N_2, \cdots, N_5$
- Relations : $r_1, r_2, \cdots, r_7$
- Types: `follows`, `posted`, `answers`

First example of a property graph



- Nodes : $N_1, N_2, \cdots, N_5$
- Relations : $r_1, r_2, \cdots, r_7$
- Types: `follows`, `posted`, `answers`
- Labels: `User`, `Admin`, `Message`

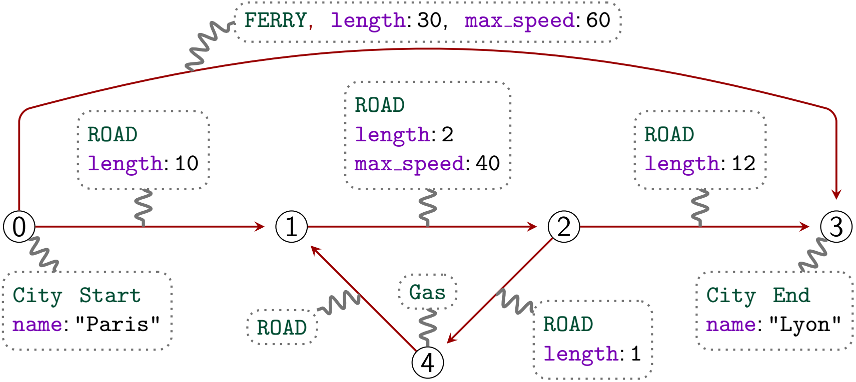
First example of a property graph



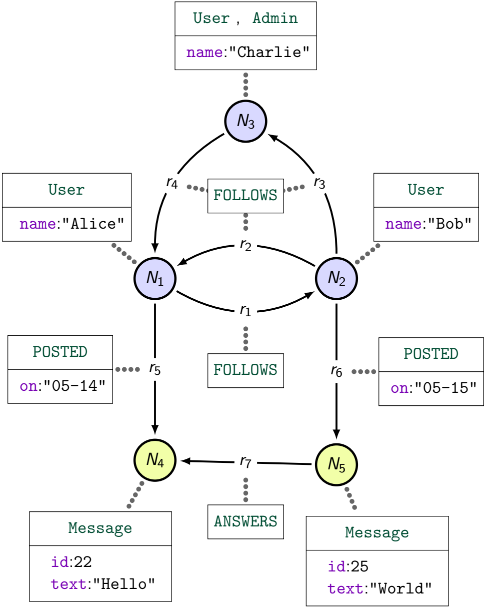
- Nodes : $N_1, N_2, \cdots, N_5$
- Relations : $r_1, r_2, \cdots, r_7$
- Types: `follows`, `posted`, `answers`
- Labels: `User`, `Admin`, `Message`
- Properties, that is Key-Value pairs:
  - `name:"Alice"`
  - `id:22`
  - `text:"Hello"`

  etc.

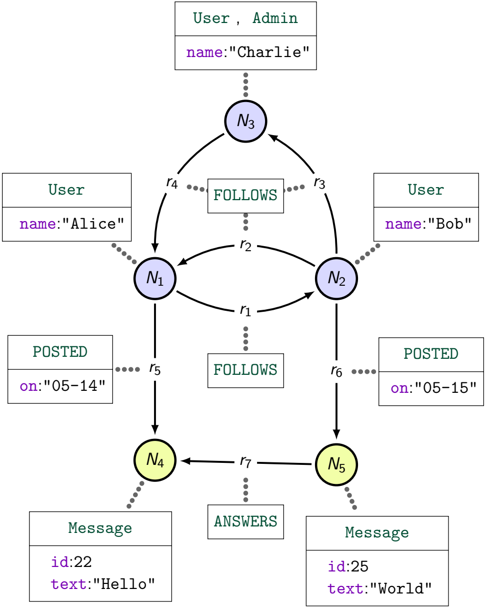
Second example of a property graph



Storing graph 1 in tables

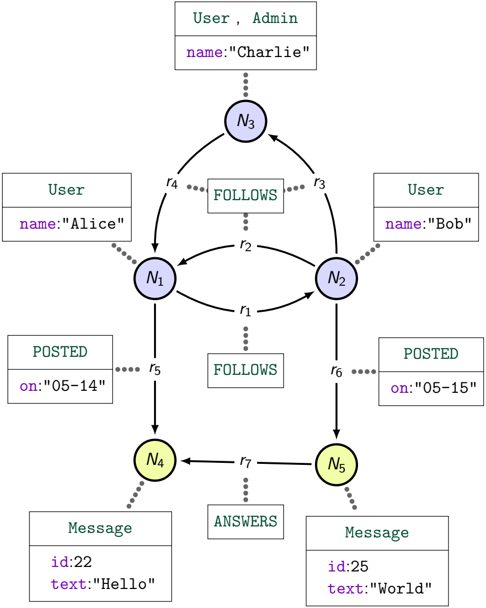


Storing graph 1 in tables



Node	**Relation**		
id	id	#src	#tgt
1	1	1	2
2	2	2	1
⋮	⋮	⋮	⋮

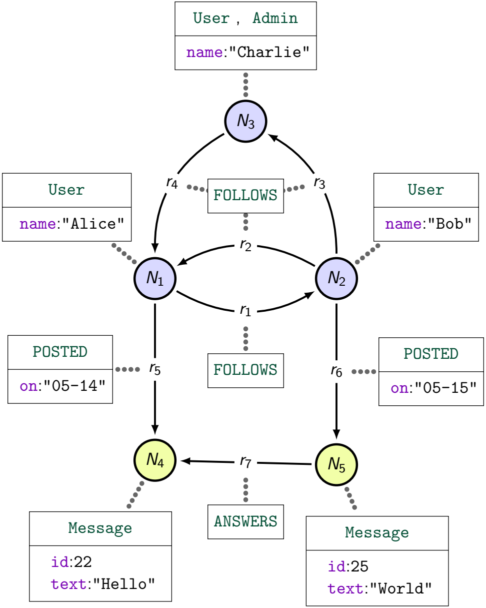
Storing graph 1 in tables



Node	Relation		
id	id	#src	#tgt
1	1	1	2
2	2	2	1
⋮	⋮	⋮	⋮

Posted	Message
#eid	#vid
5	4
6	5

Why so many tables?



Node

id
1
2
⋮

Relation

id	#src	#tgt
1	1	2
2	2	1
⋮	⋮	⋮

Posted

#eid
5
6

Message

#vid
4
5

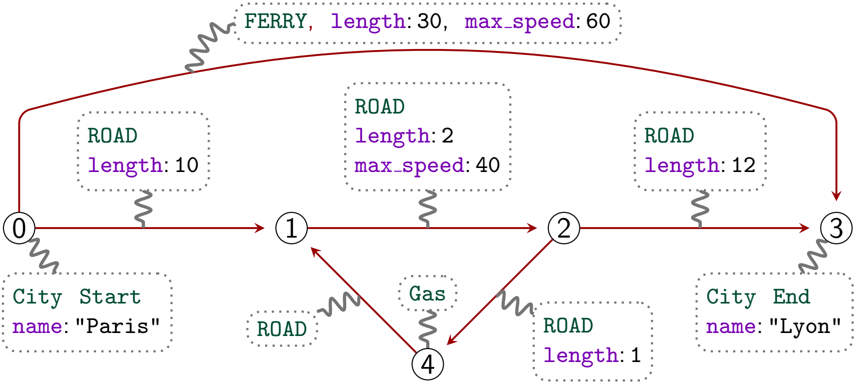
On

#eid	val
5	"05-14"
6	"05-15"

Id

#vid	val
4	22
5	25

Property graphs are very flexible



- Relations with the same type may have different property keys
- Nodes may have any number of labels and property keys

Why use property graph DBMS's?

Native storage

- Similar to "adjacency lists" : each node knows its adjacent relations
- Query answering is based on graph algorithms and not on joins
  Ex: $\mathbf{S}(\mathbf{R}+\mathbf{F})^2$, $\mathbf{S}(\mathbf{R}+\mathbf{F})^3$, $\mathbf{S}(\mathbf{R}+\mathbf{F})^*$
- Allows flexible schemas or a schema-less approach

NB: some graph DBMS do not use native storage

Why use property graph DBMS's?

Native storage

- Similar to "adjacency lists" : each node knows its adjacent relations
- Query answering is based on graph algorithms and not on joins
  Ex: $\mathbf{S}(\mathbf{R}+\mathbf{F})^2$, $\mathbf{S}(\mathbf{R}+\mathbf{F})^3$, $\mathbf{S}(\mathbf{R}+\mathbf{F})^*$
- Allows flexible schemas or a schema-less approach

NB: some graph DBMS do not use native storage

Specialized algorithms and languages

Restriction on the DM increases the liberty in the query language

- Graph notions in the core of the language (path as values)
- Graph algorithm directly available

"We never have to treat the case of non-binary relations"

NB: Relational DBMS require a graph-view (SQL/PGQ)

Part II: Neo4j, Property graphs and Cypher

2. General presentation of Cypher

A Cypher query

- queries a property graph
- returns a table

Example of Cypher query:

```
MATCH (u1)-[p1:POSTED]->(m1)
  WHERE p1.id = 22
RETURN u1.name AS uname,
       p1.on AS date,
       m1.text AS msg
```

Example Returned table

uname	date	msg
"Alice"	"05-14"	"Hello"

Generalities

A Cypher query

- queries a property graph
- returns a table
- Is a sequence of **clauses**
  (3 clauses on the right)
- Last clause is always `RETURN`

Example of Cypher query:

```
MATCH (u1)-[p1:POSTED]->(m1)
  WHERE p1.id = 22
RETURN u1.name AS uname,
       p1.on AS date,
       m1.text AS msg
```

Example Returned table

uname	date	msg
"Alice"	"05-14"	"Hello"

A Cypher query

- queries a property graph
- returns a table
- Is a sequence of **clauses** (3 clauses on the right)
- Last clause is always `RETURN`
- manipulates a working table
- uses **variables**, which refer to column names

Example of Cypher query:

```
MATCH (u1)-[p1:POSTED]->(m1)
  WHERE p1.id = 22
RETURN u1.name AS uname,
       p1.on AS date,
       m1.text AS msg
```

Example Returned table

uname	date	msg
"Alice"	"05-14"	"Hello"

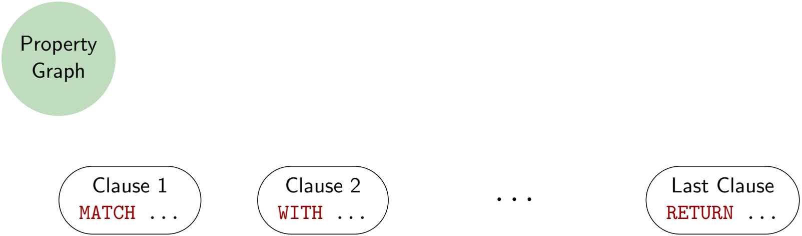
Values in Cypher

- **Values** are the elements that may appear in tables
- **Pure values** are the values with no reference to the graph
- **Property** is a key to pure values

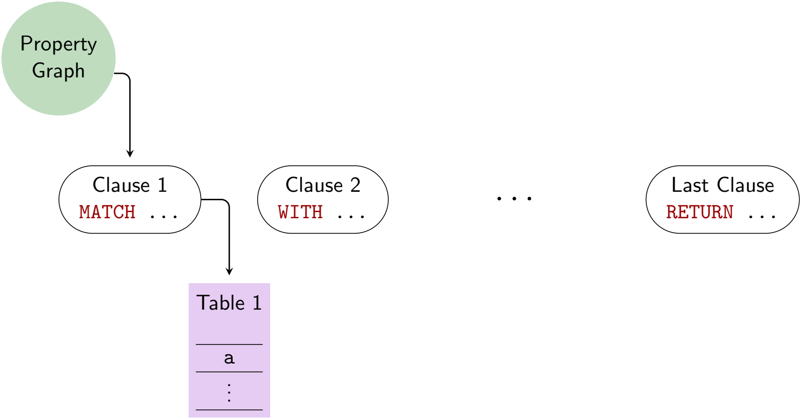
Values are

- Base values   Ex: `true`, `42`, `"NoSQL"`
- Graph elements   Ex: nodes, relations
- Paths (alternate lists of nodes and relations)
- List of values   Ex: `[1,"Hello",true,"World,`n_1`]`
- Property dictionary   Ex: `{name:"Victor", age:35}`

How evaluation works

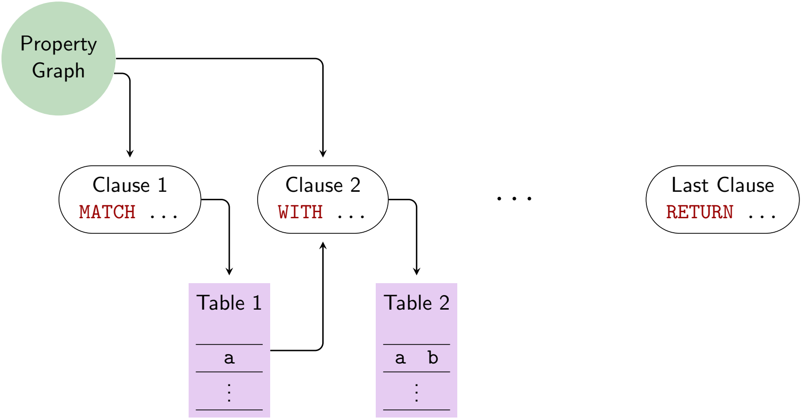


How evaluation works



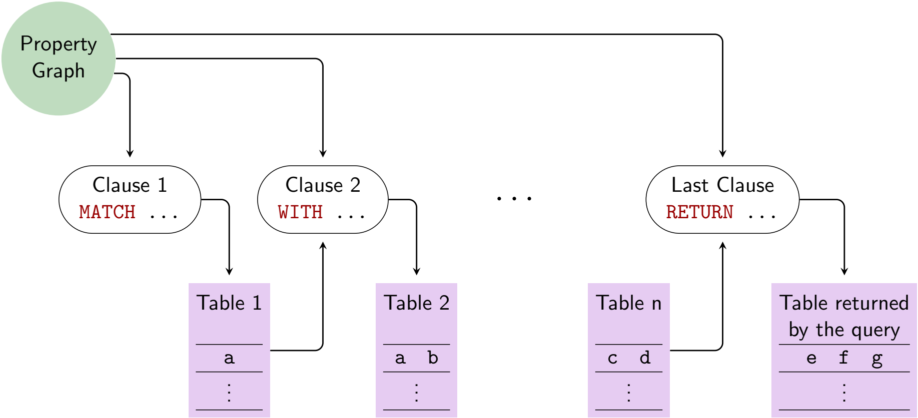
- The first clause produces a table from the property graph

How evaluation works



- The first clause produces a table from the property graph
- Subsequent clauses produces a new table from the property graph **and the prior table**

How evaluation works



- The first clause produces a table from the property graph
- Subsequent clauses produces a new table from the property graph **and the prior table**
- Until we reach the last clause, which produces the table to return

Overview of read-only Cypher

`MATCH` is for pattern matching

- RPQ-like
- Trail semantics
- Projects paths into a table
- Inner join with the input table
- The variant `OPTIONAL MATCH` does an outer join instead

`WHERE` filters rows

- Subclause of `WITH` and `MATCH`

`UNWIND` splits rows for each element in a list

`WITH` is for:

- Column manipulation (add, remove, rename, etc.)
- Aggregation
  - Vertical
  - Horizontal (`reduce`)
- Order and limit output size (`ORDER BY`, `SKIP` and `LIMIT`)

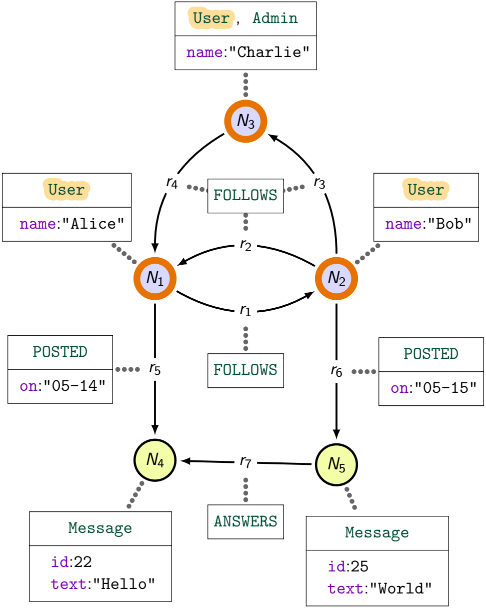
`RETURN` is a mandatory `WITH` at the end of the query

`UNION` and `UNION ALL` are for set and bag union.

Part II: Neo4j, Property graphs and Cypher

3. Pattern matching with MATCH

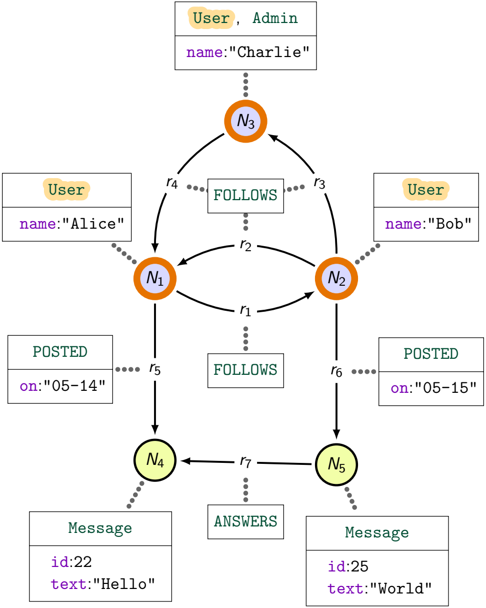
Matching nodes (1)



Query:

```
MATCH (u1:User)
```

Matching nodes (1)



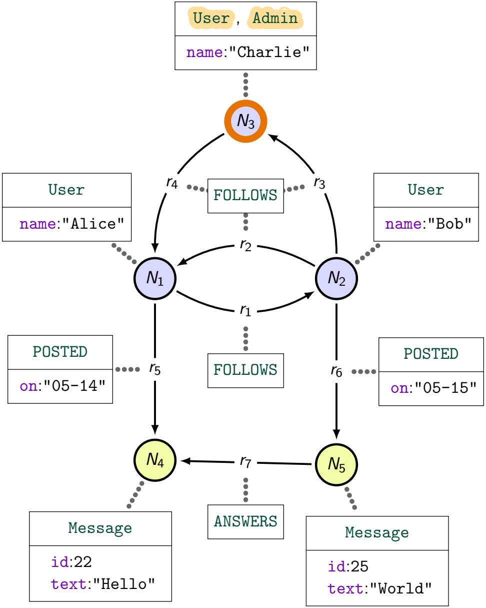
Query:

```
MATCH (u1:User)
```

Result:

u1
N_1
N_2
N_3

Matching nodes (2)



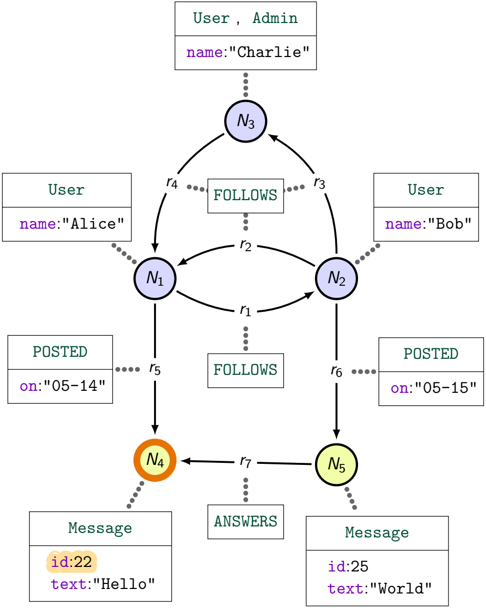
Query:

```
MATCH (u1:User:Admin)
```

Result:

u1
N_3

Matching nodes (3)



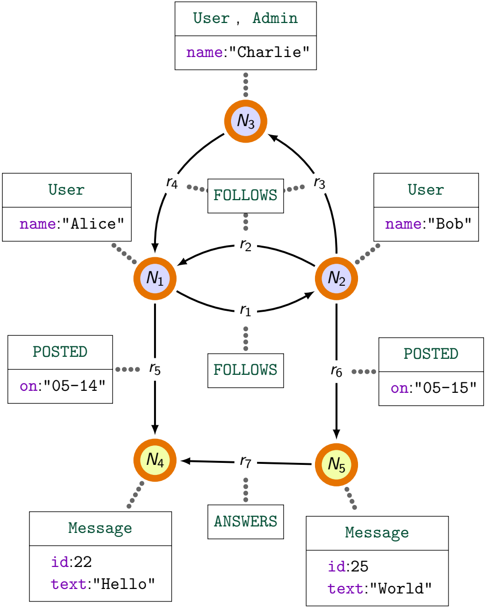
Query:

```
MATCH (u1{id:22})
```

Result:

u1
N_4

Matching nodes (4)

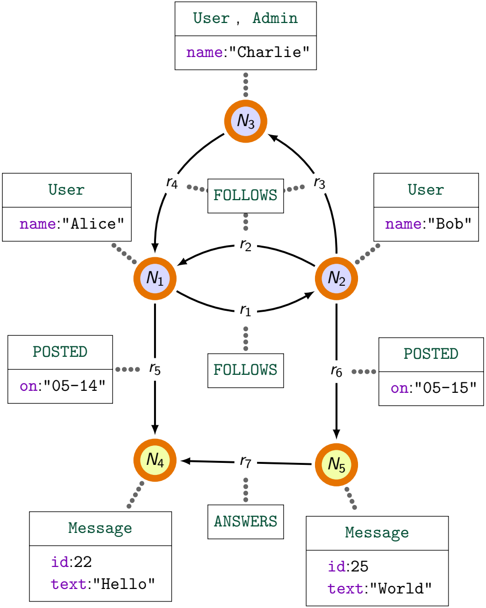


Query:

```
MATCH (u1)
```

Result:

Matching nodes (4)



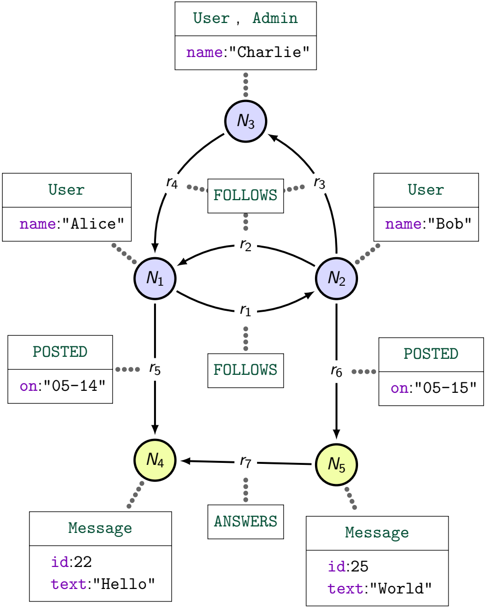
Query:

```
MATCH (u1)
```

Result:

u1
N_1
N_2
N_3
N_4
N_5

Matching relations (1)



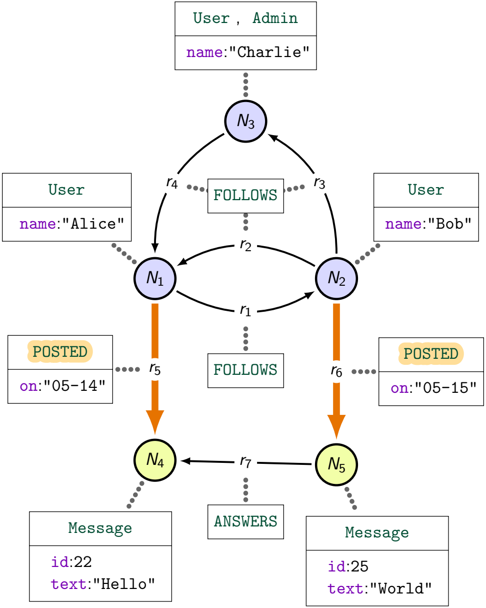
Query:

```
MATCH ()-[p1]->()
```

Result:

p1
r_1
r_2
r_3
r_4
r_5
r_6
r_7

Matching relations (2)



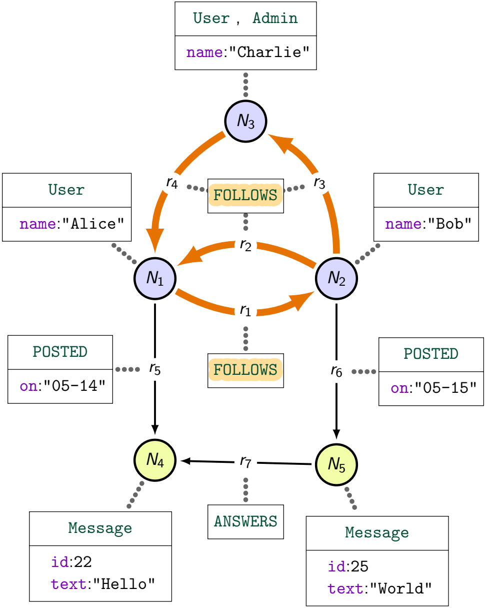
Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
```

Result:

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Matching relations (3)



Query:

```
MATCH (u1)-[:FOLLOWS]->()
```

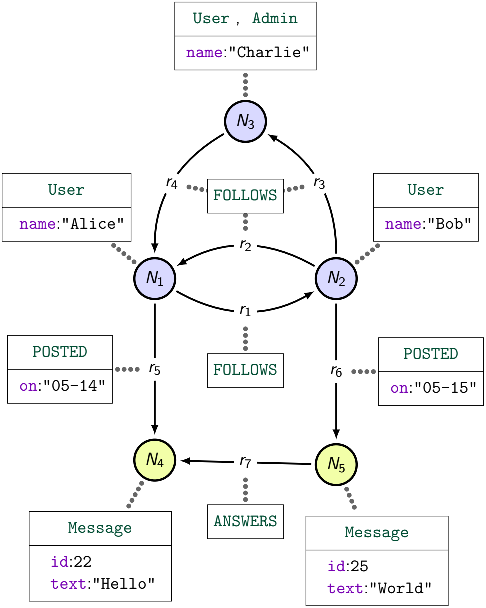
Result:

u1
N_1
N_2
N_2
N_3

Cypher has bag semantics:

N_2 has two outgoing `follows` relations $\Rightarrow$ two lines N_2

Matching joined relations (1)



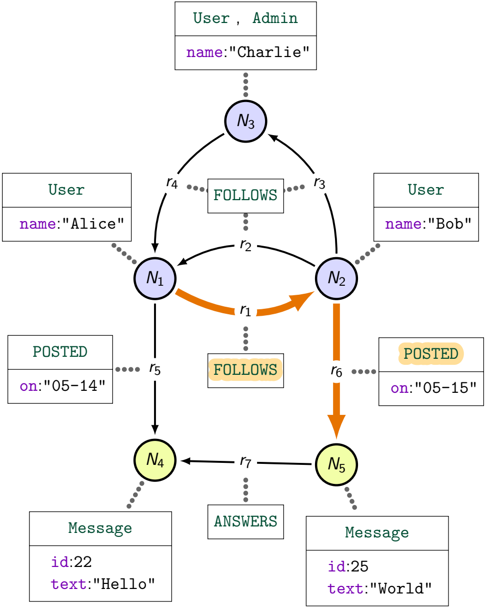
Query:

```
MATCH (u1)-[:FOLLOWS]->()
                -[:POSTED]->(m1)
```

Result:

u1	m1
N_1	N_5
N_2	N_4
N_3	N_5

Matching joined relations (1)



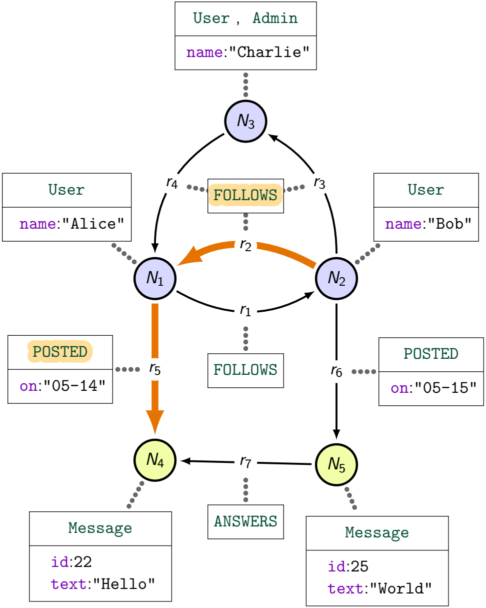
Query:

```
MATCH (u1)-[:FOLLOWS]->()
          -[:POSTED]->(m1)
```

Result:

u1	m1
N_1	N_5
N_2	N_4
N_3	N_5

Matching joined relations (1)



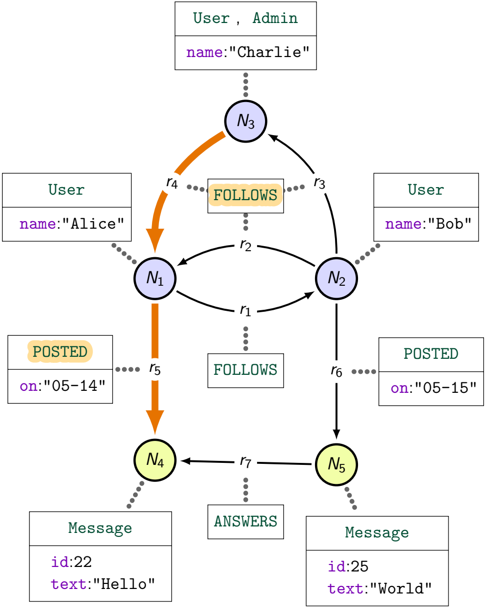
Query:

```
MATCH (u1)-[:FOLLOWS]->()
           -[:POSTED]->(m1)
```

Result:

u1	m1
N_1	N_5
N_2	N_4
N_3	N_5

Matching joined relations (1)



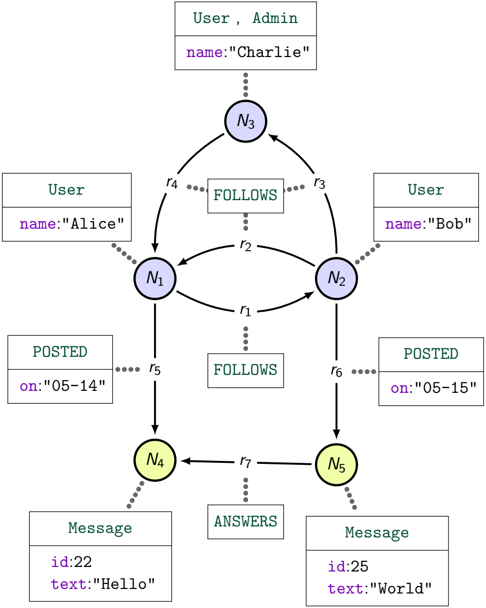
Query:

```
MATCH (u1)-[:FOLLOWS]->()
          -[:POSTED]->(m1)
```

Result:

u1	m1
N_1	N_5
N_2	N_4
N_3	N_5

Matching joined relations (2)



Query:

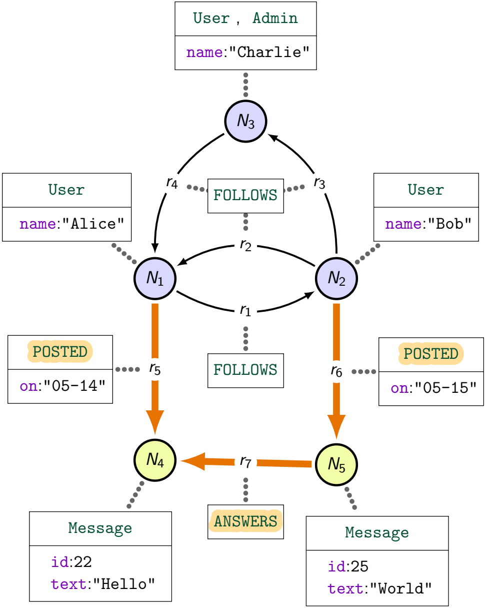
```
MATCH (u1)-[:POSTED]->()
      <-[:ANSWERS]-(m2)
      <-[:POSTED]-(u2)
```

Result:

u1	m2	m2
N_1	N_5	N_2

⚠ Cypher allows backward relations

Matching joined relations (2)



Query:

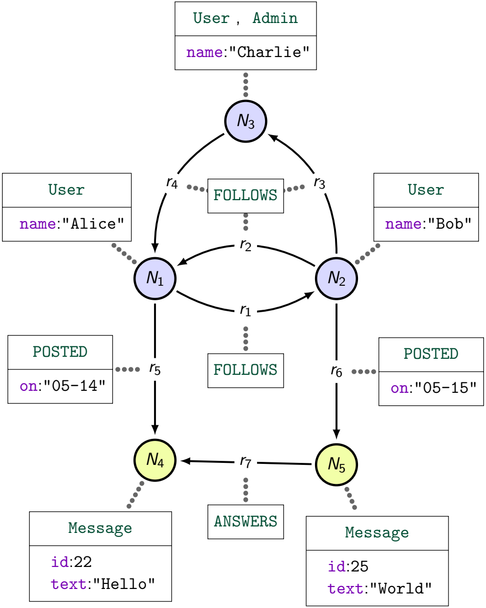
```
MATCH (u1)-[:POSTED]->()
      <-[:ANSWERS]-(m2)
      <-[:POSTED]-(u2)
```

Result:

u1	m2	m2
N_1	N_5	N_2

⚠ Cypher allows backward relations

Implicit join on variable reuse



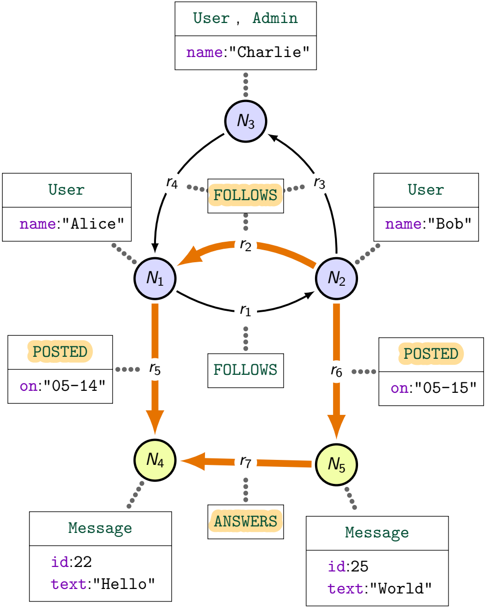
Query:

```
MATCH (u1)-[:POSTED]->()
          <-[:ANSWERS]-(m2)
          <-[:POSTED]-(u2)
          -[:FOLLOWS]->(u1)
```

Result:

u1	u2	m2
N_1	N_5	N_2

Implicit join on variable reuse



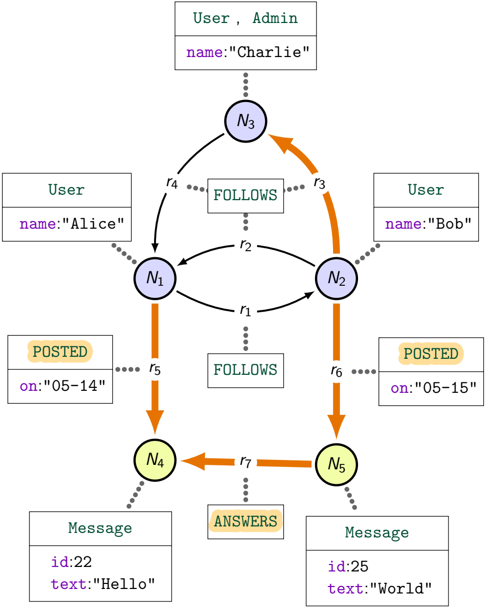
Query:

```
MATCH (u1)-[:POSTED]->()
          <-[:ANSWERS]-(m2)
          <-[:POSTED]-(u2)
          -[:FOLLOWS]->(u1)
```

Result:

u1	u2	m2
N_1	N_5	N_2

Implicit join on variable reuse



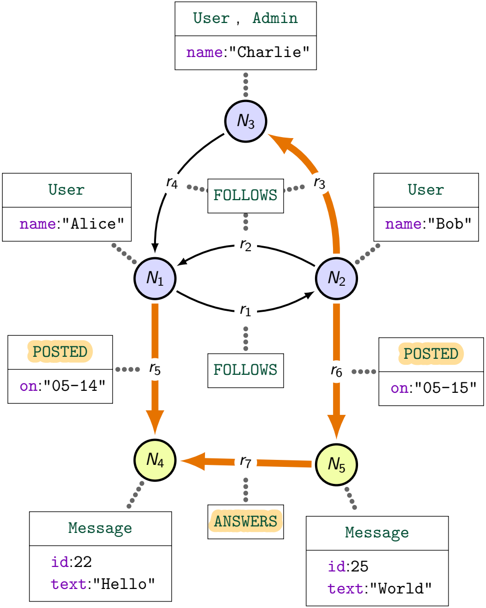
Query:

```
MATCH (u1)-[:POSTED]->()
         <-[:ANSWERS]-(m2)
         <-[:POSTED]-(u2)
         -[:FOLLOWS]->(u1)
```

Result:

u1	u2	m2
N_1	N_5	N_2

Implicit join on variable reuse



Query:

```
MATCH (u1)-[:POSTED]->()
        <-[:ANSWERS]-(m2)
        <-[:POSTED]-(u2)
        -[:FOLLOWS]->(u1)
```

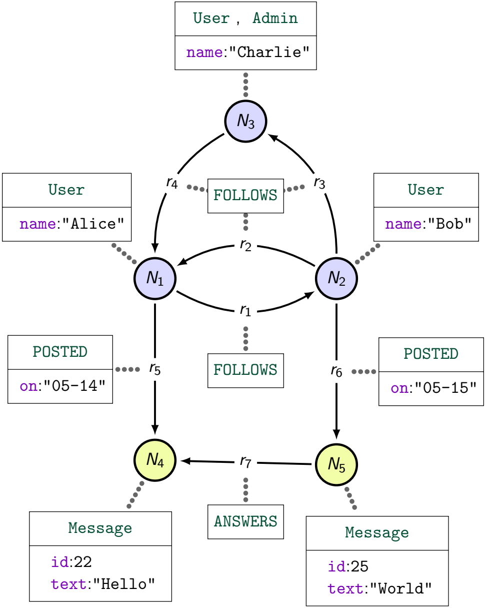
Result:

u1	u2	m2
N_1	N_5	N_2

This path is invalid: two different nodes for u1.

Variable reuse $\Longrightarrow$ equality

Matching paths (1)



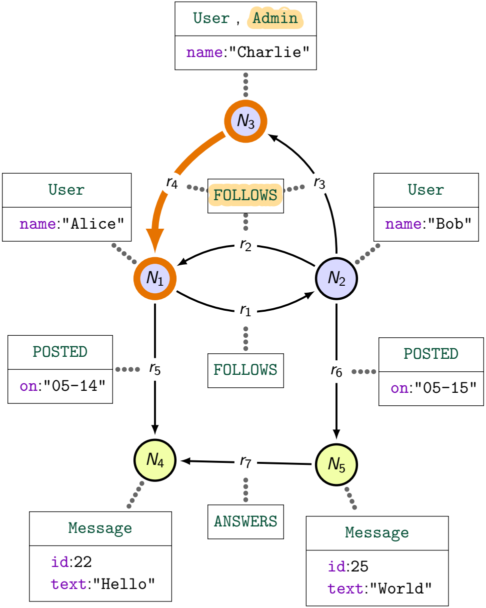
Query:

```
MATCH (u1:Admin)
      -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



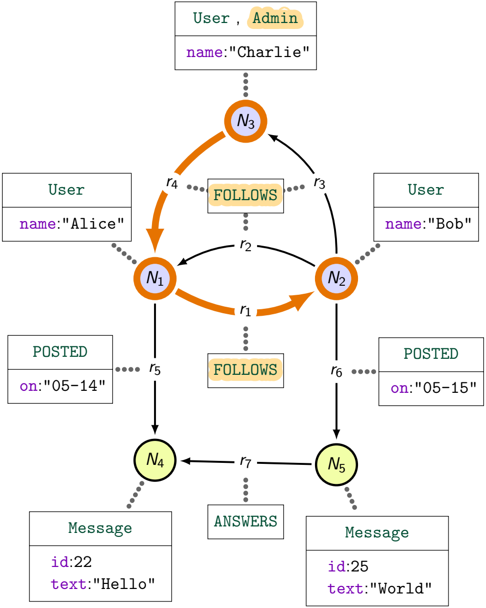
Query:

```
MATCH (u1:Admin)
        -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



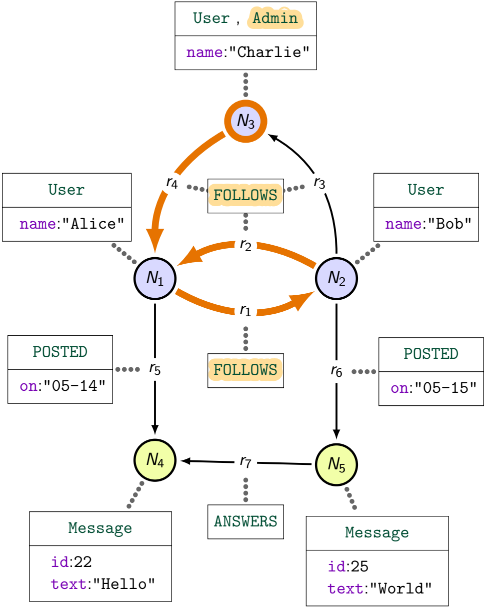
Query:

```
MATCH (u1:Admin)
        -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



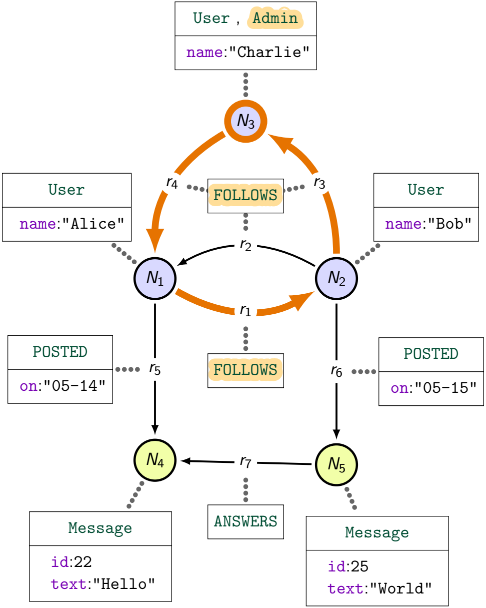
Query:

```
MATCH (u1:Admin)
        -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



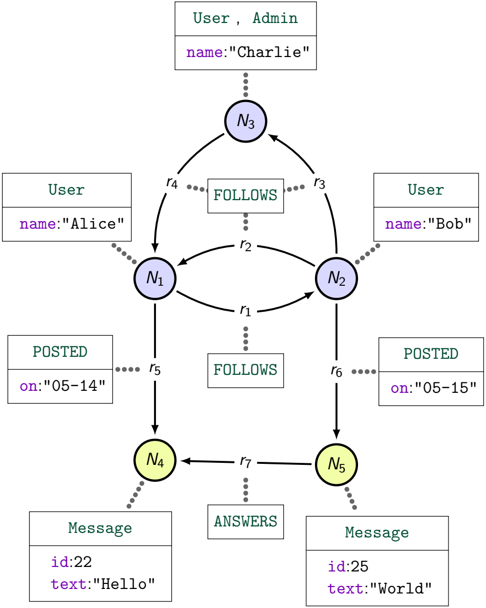
Query:

```
MATCH (u1:Admin)
          -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



Query:

```
MATCH (u1:Admin)
        -[l1:FOLLOWS*]->(m1)
```

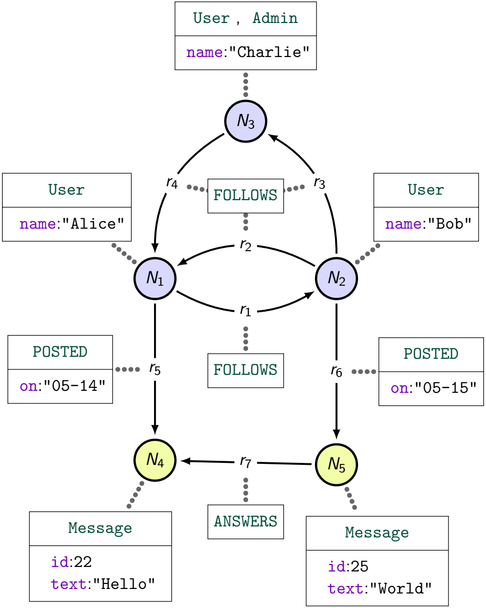
Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Cypher uses **trail semantics**.

In Cypher the Kleene star means **one or more**.

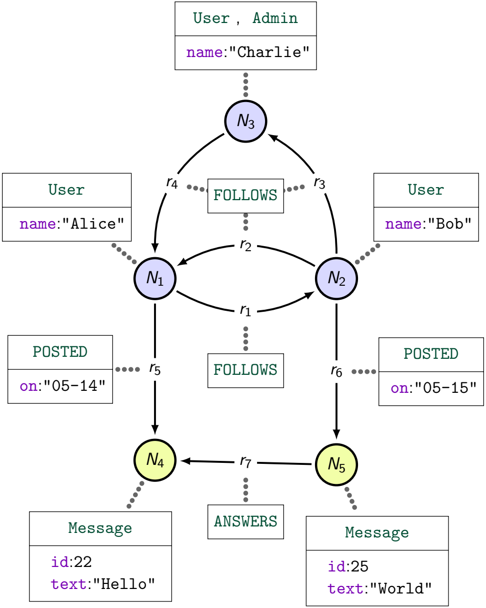
An interesting query



Query:

```
MATCH (u2:)-[:FOLLOWS]->(u1)
                    <-[:FOLLOWS]-(u3)
```

An interesting query



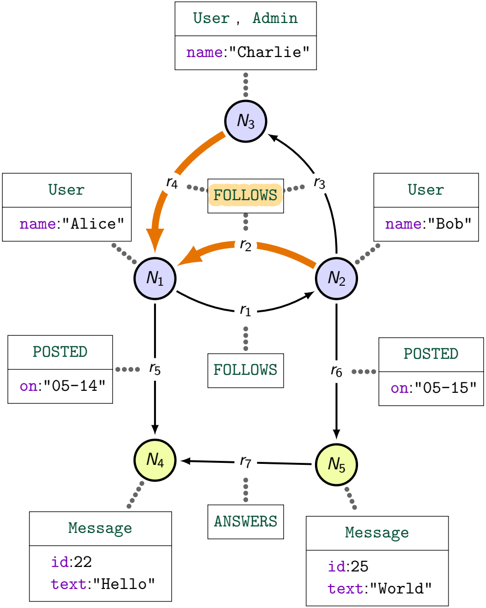
Query:

```
MATCH (u2:)-[:FOLLOWS]->(u1)
                  <-[:FOLLOWS]-(u3)
```

Result:

u2	u1	u3
N_3	N_1	N_2
N_2	N_1	N_3

- Line 1: $N_3 \xrightarrow{r_4} N_1 \xleftarrow{r_2} N_2$
- Line 2: $N_2 \xrightarrow{r_2} N_1 \xleftarrow{r_4} N_3$
- No (N_3, N_1, N_3) due to trail semantics



Query:

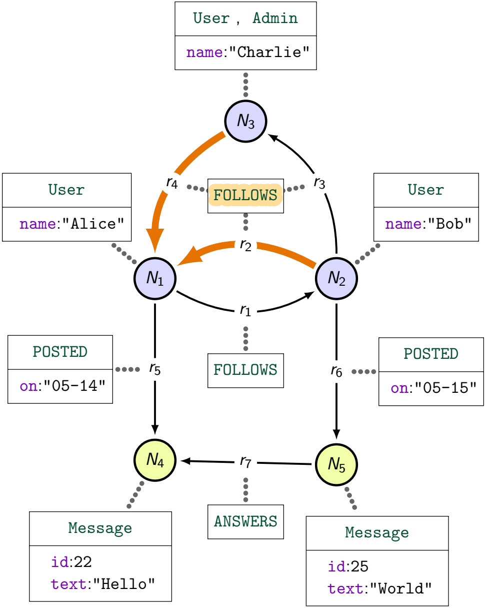
```
MATCH (u2:)-[:FOLLOWS]->(u1)
                <-[:FOLLOWS]-(u3)
```

Result:

u2	u1	u3
N_3	N_1	N_2
N_2	N_1	N_3

- Line 1: $N_3 \xrightarrow{r_4} N_1 \xleftarrow{r_2} N_2$
- Line 2: $N_2 \xrightarrow{r_2} N_1 \xleftarrow{r_4} N_3$
- No (N_3, N_1, N_3) due to trail semantics

An interesting query



Query:

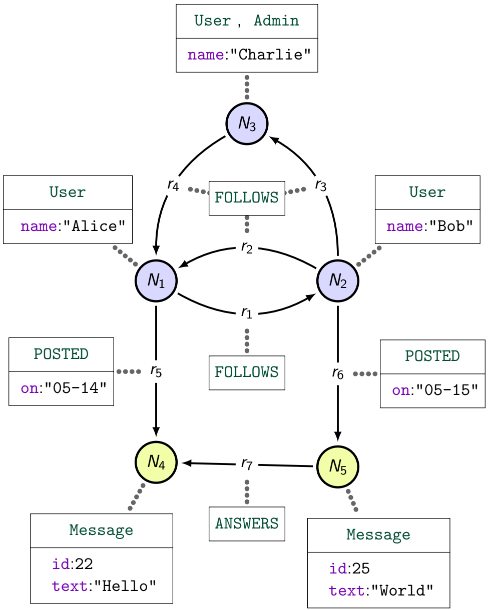
```
MATCH (u2:)-[:FOLLOWS]->(u1)
                     <-[:FOLLOWS]-(u3)
```

Result:

u2	u1	u3
N_3	N_1	N_2
N_2	N_1	N_3

- Line 1: $N_3 \xrightarrow{r_4} N_1 \xleftarrow{r_2} N_2$
- Line 2: $N_2 \xrightarrow{r_2} N_1 \xleftarrow{r_4} N_3$
- No (N_3, N_1, N_3) due to trail semantics

Matching paths (2)



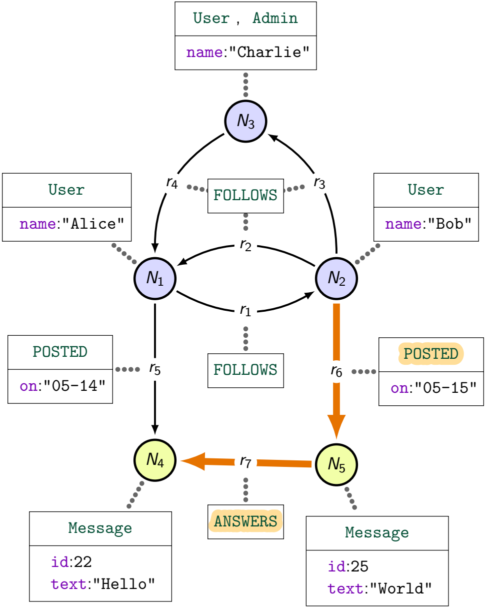
Query:

```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5

Matching paths (2)



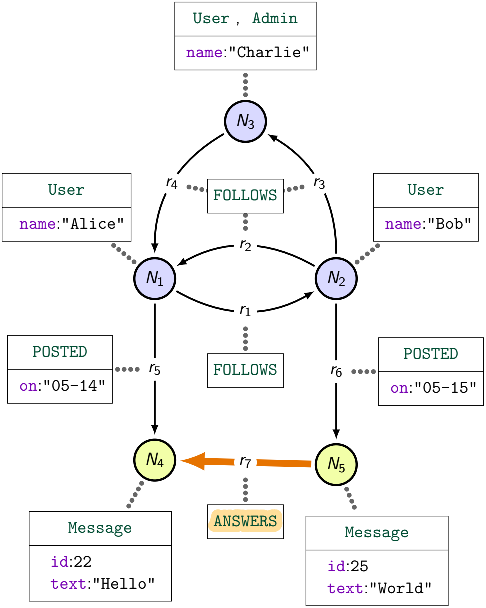
Query:

```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5

Matching paths (2)



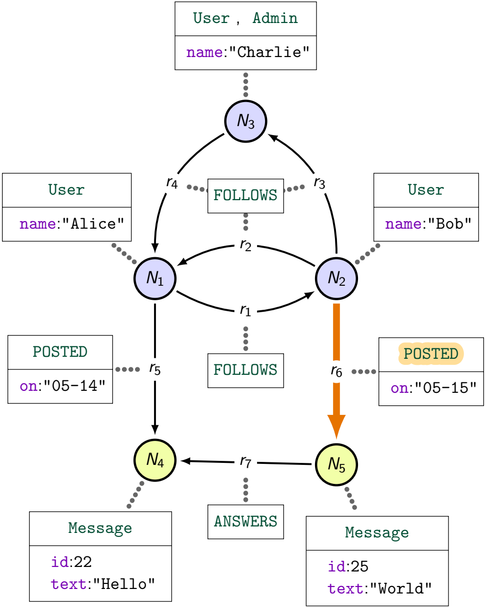
Query:

```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5

Matching paths (2)



Query:

```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5

Recap of MATCH

MATCH allows RPQ-like pattern-matching

- Computes paths
- Uses trail semantics to keep the output finite
- Project paths on variables

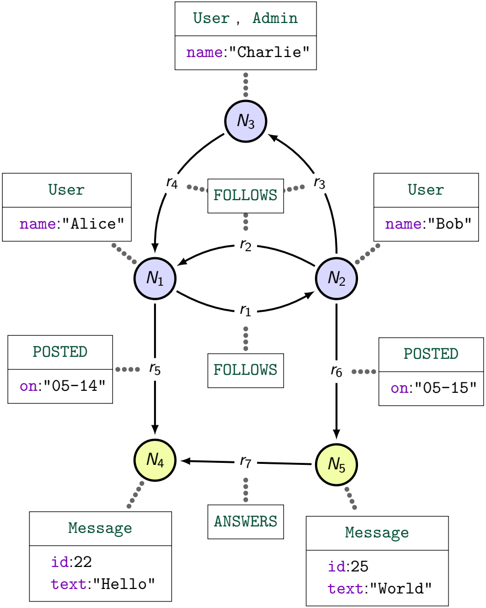
MATCH does not allow the full extent of regular expressions

- Only disjunction of atoms under star Ex: $(\mathbf{R}^*\mathbf{G})^*$ and $(\mathbf{RR})^*$
- Disjunction only for atoms Ex. $\mathbf{R}^* + \mathbf{F}$ and $\mathbf{RR} + \mathbf{FF}$)

MATCH goes beyond RPQs

- Matching against properties Ex: `MATCH ({id:22})`
- Taking relation backward Ex: `MATCH ()<-[e]-()`
- Implicit join on variable reuse Ex: `MATCH (a)<-[*]-(a)`

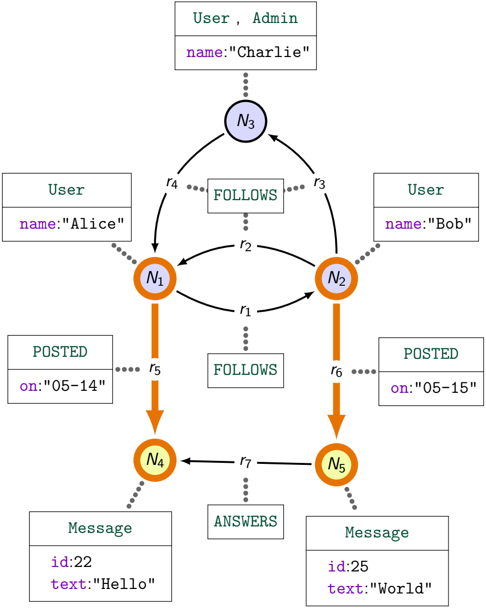
Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
              -[:FOLLOWS]->(u3)
```

Sequence of MATCH clauses



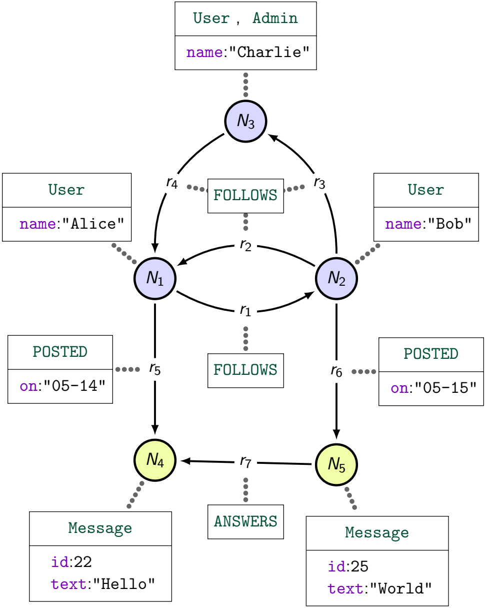
Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
            -[:FOLLOWS]->(u3)
```

Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                    -[:FOLLOWS]->(u3)
```

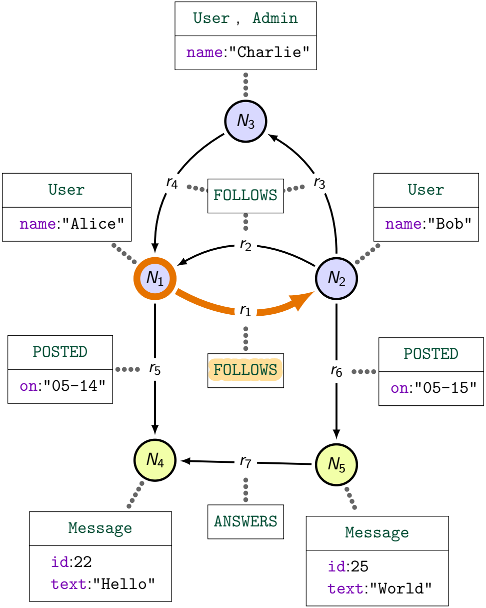
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
N_1	N_4	·	·
N_2	N_5	·	·

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
              -[:FOLLOWS]->(u3)
```

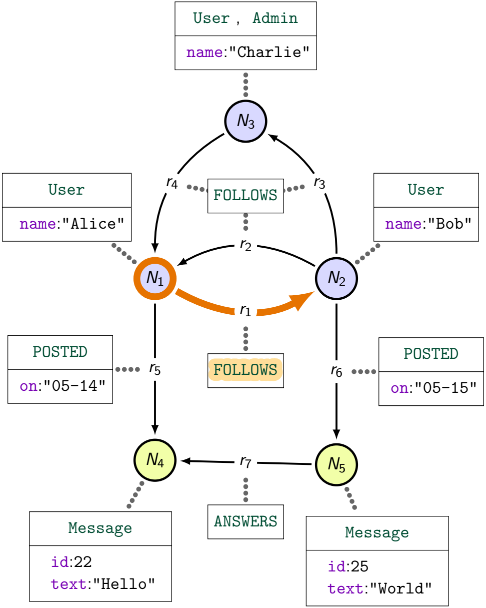
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
N_1	N_4	·	·
N_2	N_5	·	·

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                    -[:FOLLOWS]->(u3)
```

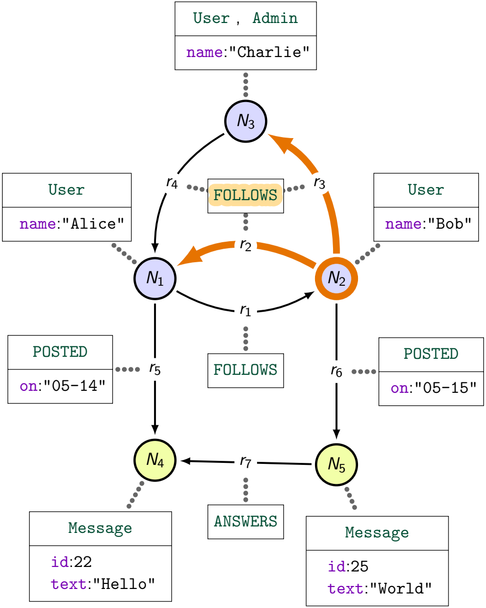
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
~~N_1~~	~~N_4~~		
N_2	N_5	·	·

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                 -[:FOLLOWS]->(u3)
```

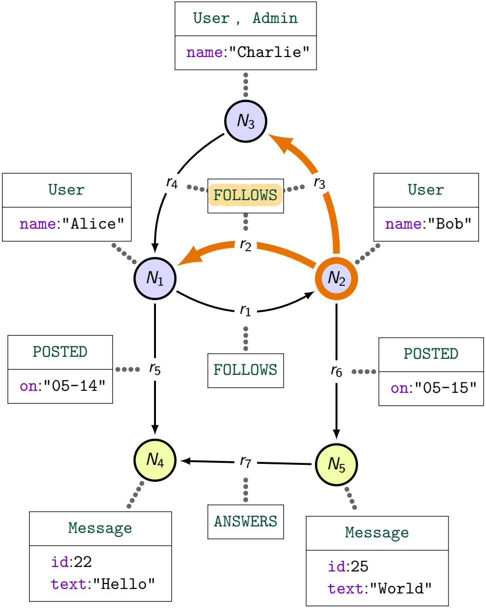
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
~~N_1~~	~~N_4~~		
N_2	N_5	·	·

Sequence of MATCH clauses



Query:

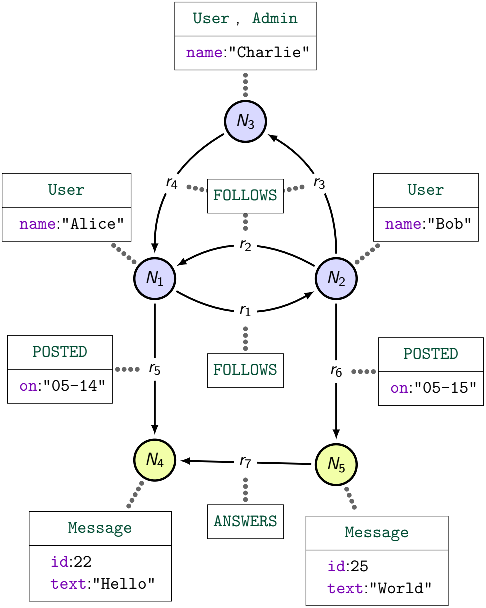
```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                  -[:FOLLOWS]->(u3)
```

Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
N_2	N_5	N_1	N_3
N_2	N_5	N_3	N_1



The two following queries compute similar thing:

```
MATCH (a)⟨pat1⟩(b)⟨pat2⟩(c)
```

```
MATCH (a)⟨pat1⟩(b)
MATCH (b)⟨pat2⟩(c)
```

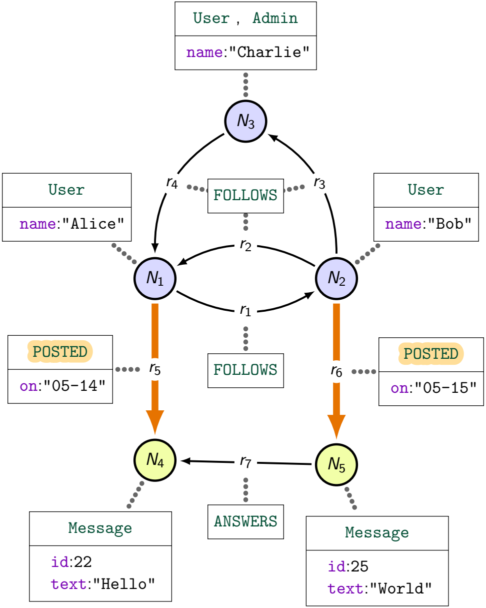
1 Compute their answer for
$\langle pat_1 \rangle =$ `-[:FOLLOWS]->`
$\langle pat_1 \rangle =$ `-[:POSTED]->`

2 Can you find patterns $\langle pat_1 \rangle$ and $\langle pat_2 \rangle$ for which their answer is different?

Part II: Neo4j, Property graphs and Cypher

4. Usage of WITH (or RETURN)

Column manipulation



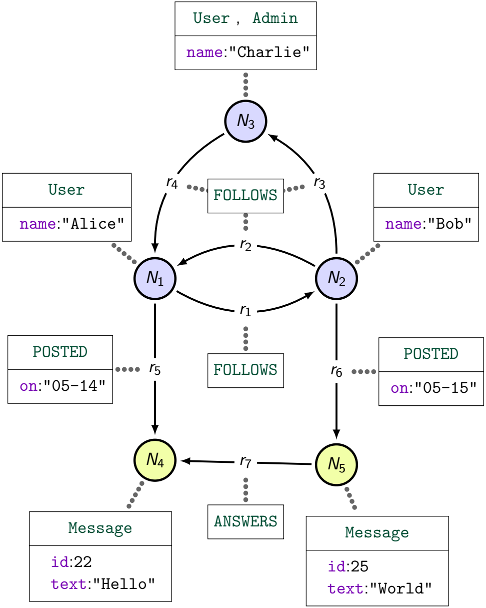
Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

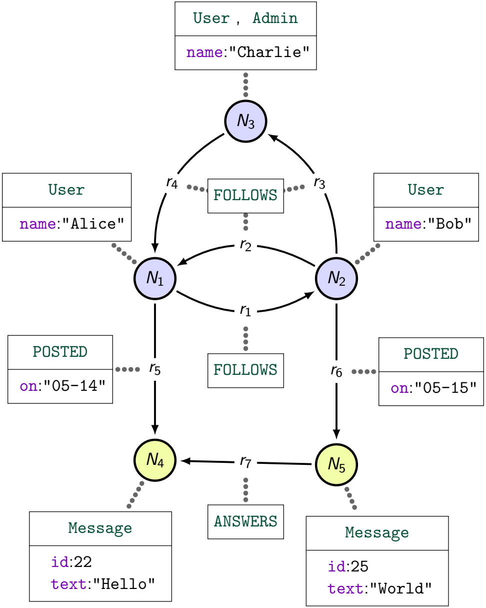
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WITH clause

u1	p1	t1
N_1	r_5	
N_2	r_6	

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

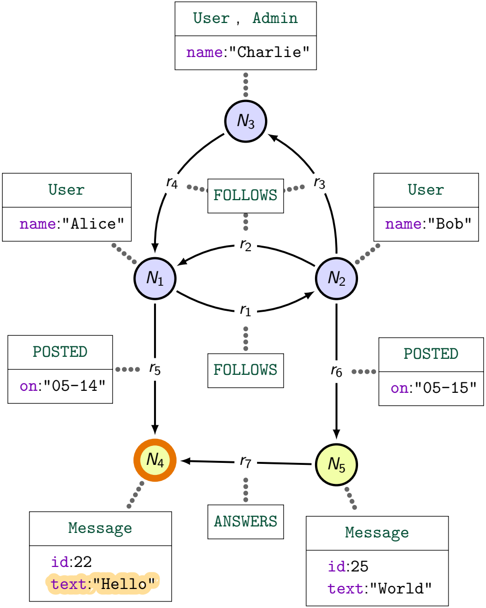
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WITH clause

u1	p1	t1
N_1	r_5	
N_2	r_6	

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

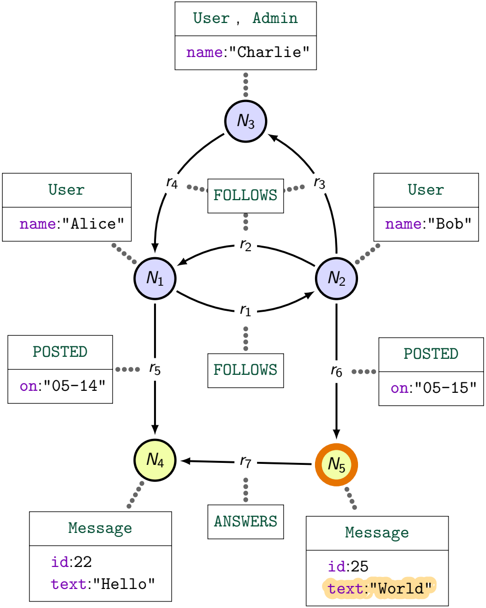
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WITH clause

u1	p1	t1
N_1	r_5	"Hello"
N_2	r_6	

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

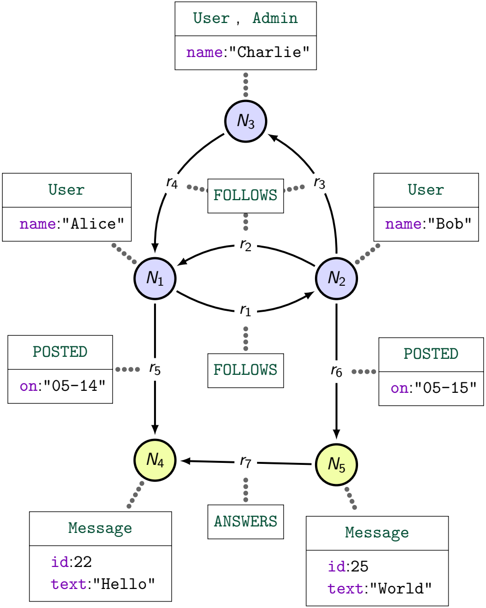
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WITH clause

u1	p1	t1
N_1	r_5	"Hello"
N_2	r_6	"World"

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

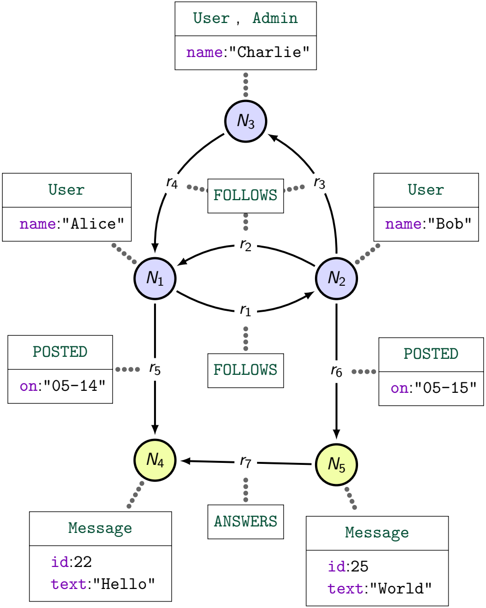
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Final result

u1	p1	t1
N_1	r_5	"Hello"
N_2	r_6	"World"

Elimination of duplicate rows



Query:

```
MATCH (u1)-[:FOLLOWS]->()
WITH DISTINCT u1
```

After MATCH:

u1
N_1
N_2
N_2
N_3

After WITH:

u1
N_1
N_2
N_3

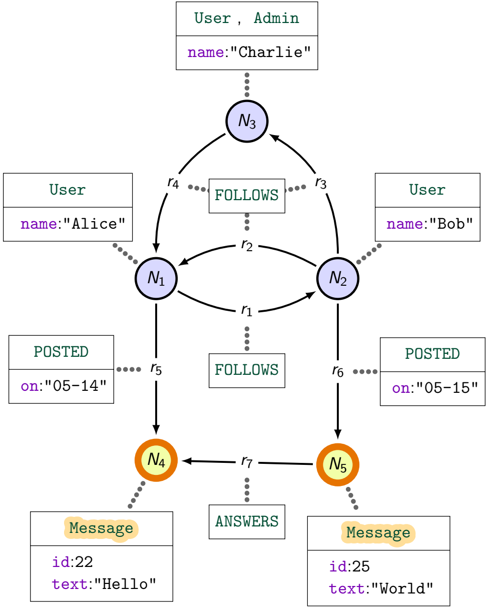
Vertical Aggregation

WITH ⟨*columns*⟩, ⟨*aggr*⟩(⟨*expr*⟩)

Grouping is implicit: every variable used in ⟨*columns*⟩ is used for grouping

⟨*aggr*⟩ in a built-int **aggregation function**, that is, a function from list to a single value.

Example: `count`, `sum`, `min`, `collect`, etc.



Query:

```
MATCH (m1:Message)
WITH count(m1) AS c
```

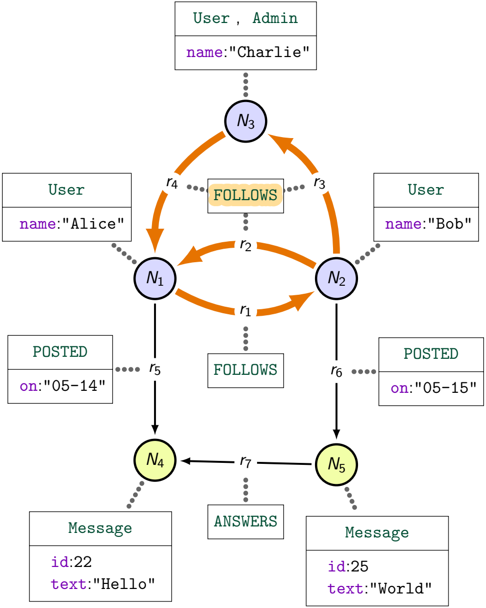
After MATCH:

m1
N_4
N_5

After WITH:

c
2

Collecting names of followers



Query:

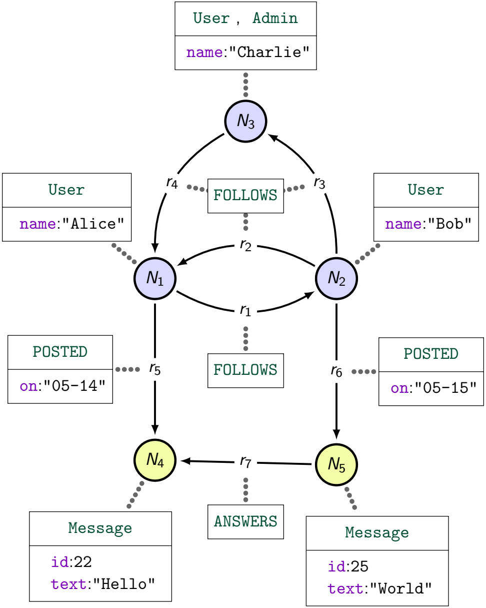
```
MATCH (u1)<-[:FOLLOWS]-(u2)
WITH u1, collect(u2.name) AS n
```

Result after WITH:

u1	n
N_1	["Bob","Charlie"]
N_2	["Alice"]
N_3	["Bob"]

⚠ Grouping by u1

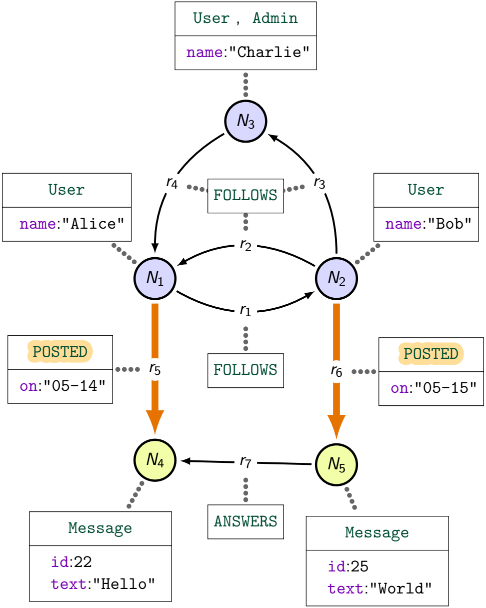
Exercice: what does this compute?



Query:

```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
          {on:d}]->(m1)
WITH m1.text as txt
```

Exercice: what does this compute?

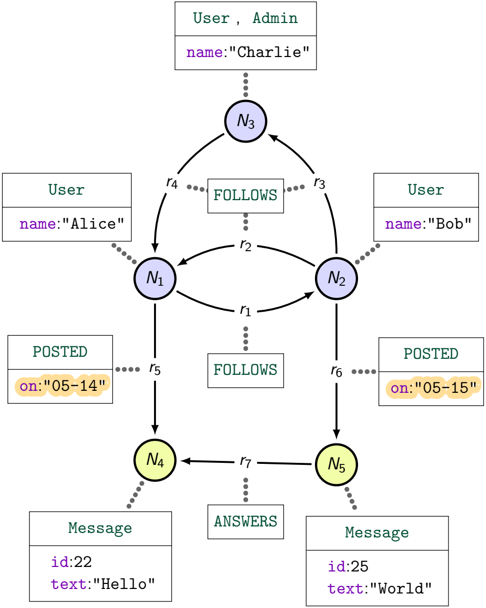


Query:

```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
              {on:d}]->(m1)
WITH m1.text as txt
```

e
r_5
r_6

Exercice: what does this compute?



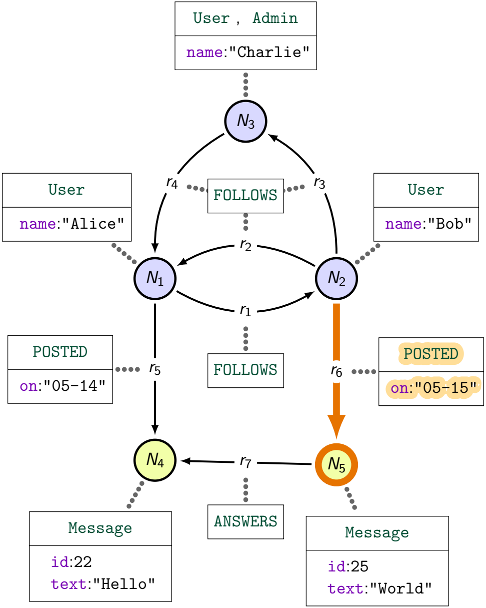
Query:

```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
              {on:d}]->(m1)
WITH m1.text as txt
```

e
r_5
r_6

d
"05-15"

Exercice: what does this compute?



Query:

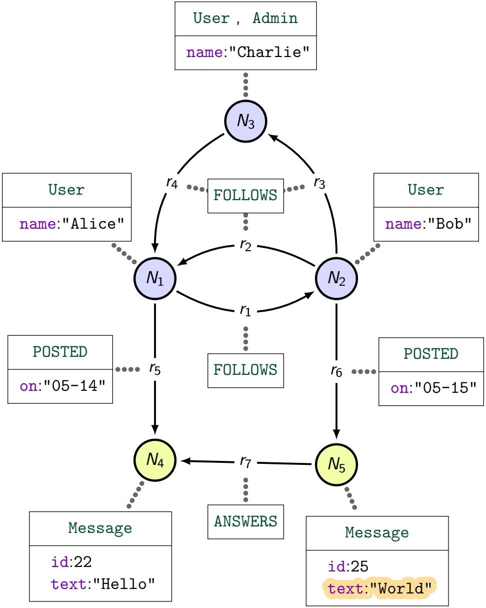
```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
              {on:d}]->(m1)
WITH m1.text as txt
```

e
r_5
r_6

d
"05-15"

d	m1
"05-15"	N_5

Exercice: what does this compute?



Query:

```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
              {on:d}]->(m1)
WITH m1.text as txt
```

e
r_5
r_6

d
"05-15"

d	m1
"05-15"	N_5

txt
"World"

Horizontal aggregation

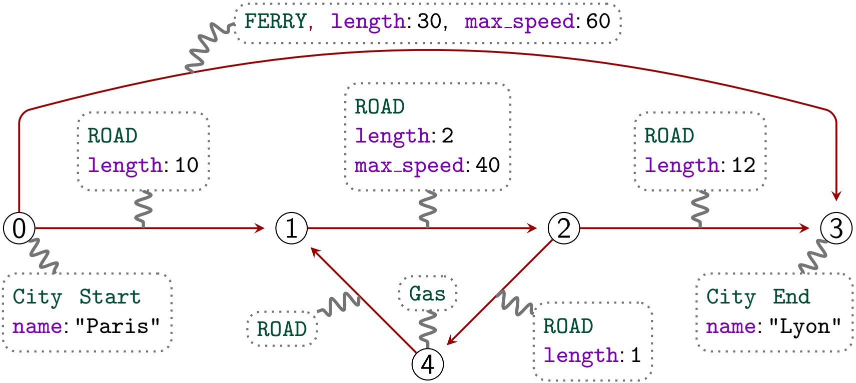
Syntax

```
reduce(⟨acc⟩ = ⟨init⟩, ⟨var⟩ IN ⟨list⟩ | ⟨update⟩)
```

Equivalent to the following pseudo code

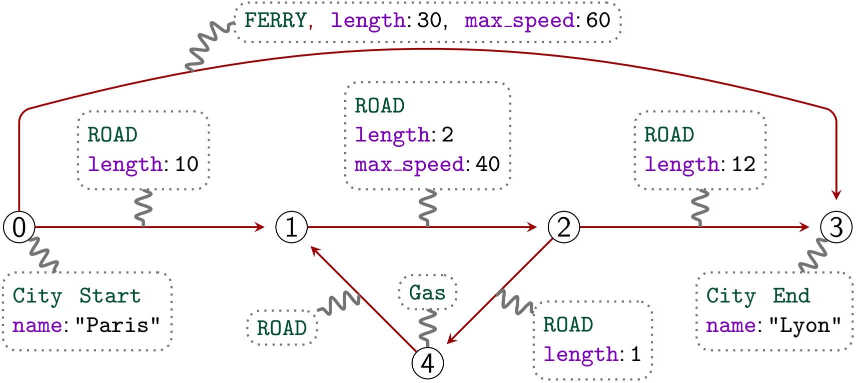
```
⟨acc⟩ := ⟨init⟩
for ⟨var⟩ in ⟨list⟩:
    ⟨acc⟩ := ⟨update⟩
```

Computing the length of a path




```
MATCH (:Start)-[e:ROAD|FERRY*]->(:End)
WITH reduce(acc=0, x IN e | acc+x.length) AS l
```

Computing the duration of a path




```
MATCH (:Start)-[e:ROAD|FERRY*]->(:End)
WITH reduce(acc = 0, x IN e
                | acc + x.length*coalesce(x.max_speed,80)) AS d
```

Why “horizontal” and “vertical” aggregation

Part II: Neo4j, Property graphs and Cypher

5. Subclauses of `MATCH` and/or `WITH`

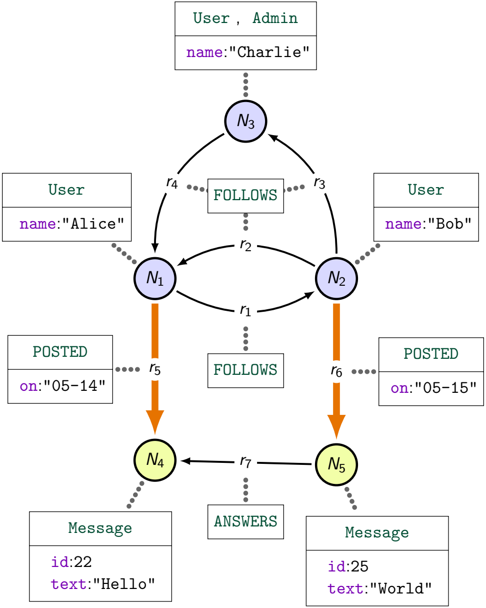
Filtering rows with `WHERE` (1)

Syntax

`MATCH` … `WHERE` ⟨*condition*⟩
or
`WITH` … `WHERE` ⟨*condition*⟩

Remove from the table computed by `MATCH` or `WHERE` the row that make ⟨*condition*⟩ false

Filtering rows with WHERE (2)



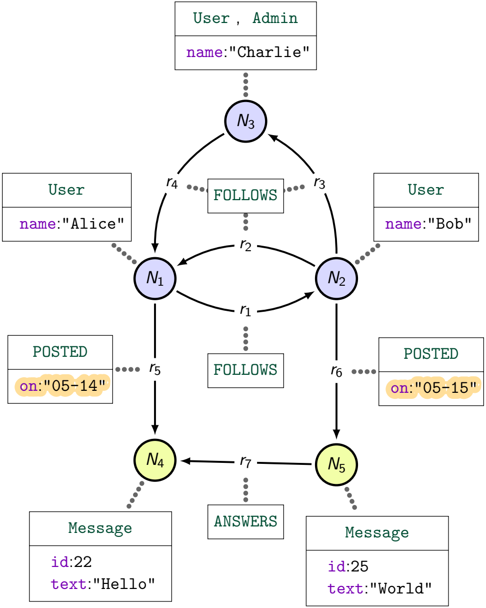
Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WHERE p1.on > "05-14"
```

After the WITH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Filtering rows with WHERE (2)



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WHERE p1.on > "05-14"
```

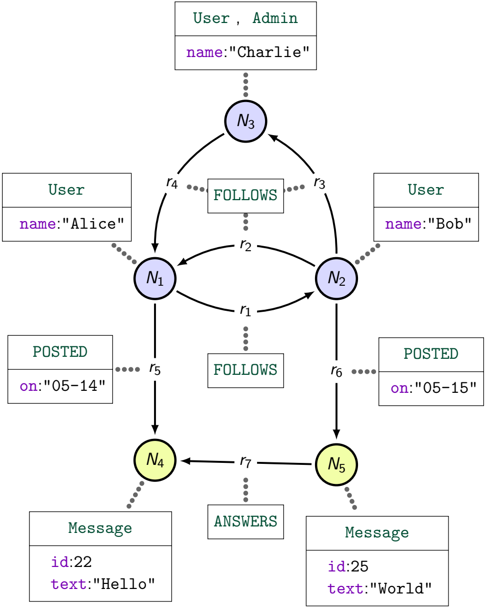
After the WITH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WHERE clause

u1	p1	m1
N_1	r_5	N_4
~~N_2~~	~~r_6~~	~~N_5~~

Filtering rows with WHERE (2)



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WHERE p1.on > "05-14"
```

After the WITH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Final result

u1	p1	m1
N_1	r_5	N_4

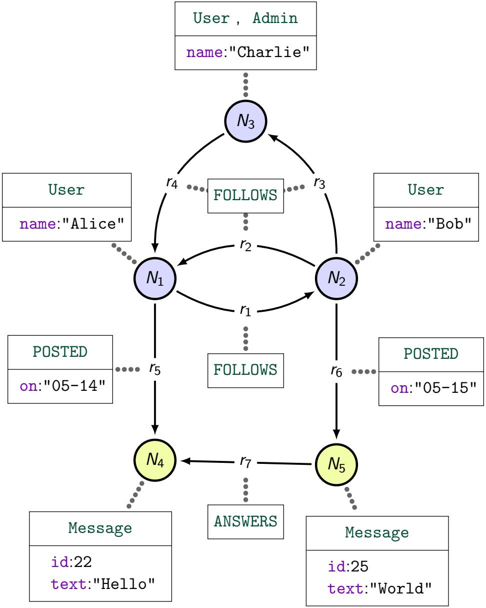
ORDER BY, LIMIT and SKIP

Syntax

WITH … $\underbrace{\texttt{ORDER BY}\ \langle \textit{oexpr}_1 \rangle\ \overbrace{\texttt{DESC}}^{\text{optional}}, \ldots}_{\text{optional}}$ $\underbrace{\texttt{SKIP}\ \langle \textit{sexpr} \rangle}_{\text{optional}}$ $\underbrace{\texttt{LIMIT}\ \langle \textit{lexpr} \rangle}_{\text{optional}}$

- Order the table by $\langle \textit{oexpr}_1 \rangle$
  - Ties are broken by the value of $\langle \textit{oexpr}_2 \rangle$, remaining ties are broken by $\langle \textit{oexpr}_3 \rangle$, etc
  - DESC means the order is descending.
  - ⚠ We might end up with ties → Nondeterminism
- Then, remove the first $\langle \textit{sexpr} \rangle$ rows
- Then, keep the first $\langle \textit{lexpr} \rangle$ rows, at most

Compute the User with the most followers

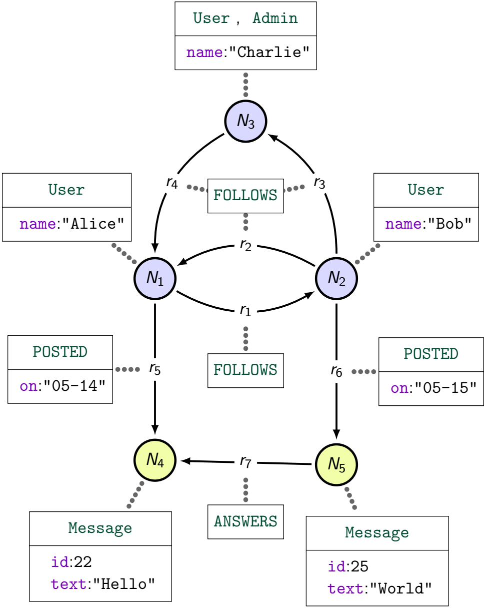


Query:

```
MATCH (u1)<-[:FOLLOWS]-(u2)
WITH u1, count(b) AS c
  ORDER BY c
  LIMIT 1 DESC
```

u1	c
N_1	2

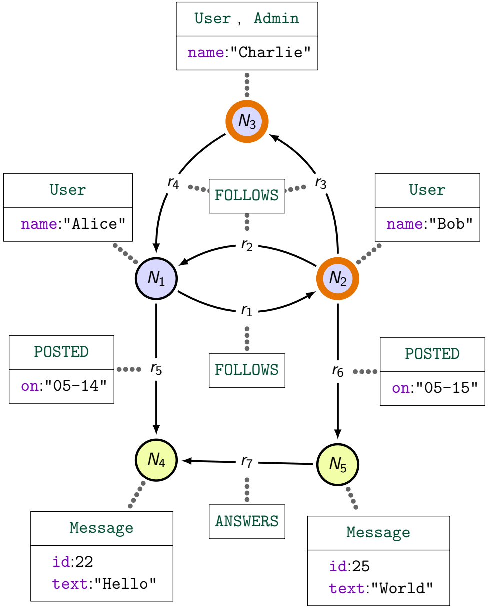
Compute the two User with the most followers



Query:

```
MATCH (u1)<-[:FOLLOWS]-(u2)
WITH u1, count(b) AS c
  ORDER BY c DESC
  LIMIT 2
```

Compute the two User with the most followers



Query:

```
MATCH (u1)<-[:FOLLOWS]-(u2)
WITH u1, count(b) AS c
  ORDER BY c DESC
  LIMIT 2
```

Since Charlie and Bob both have 1 follower, the final table is either:

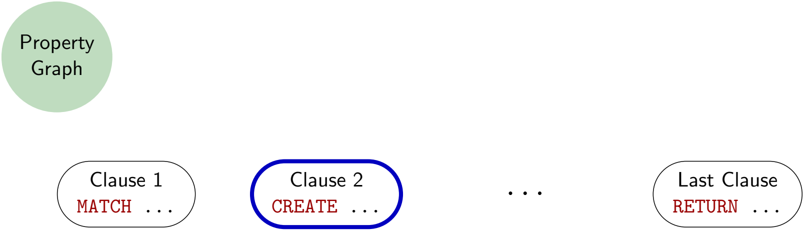
u1	c
N_1	2
N_2	1

u1	c
N_1	2
N_3	1

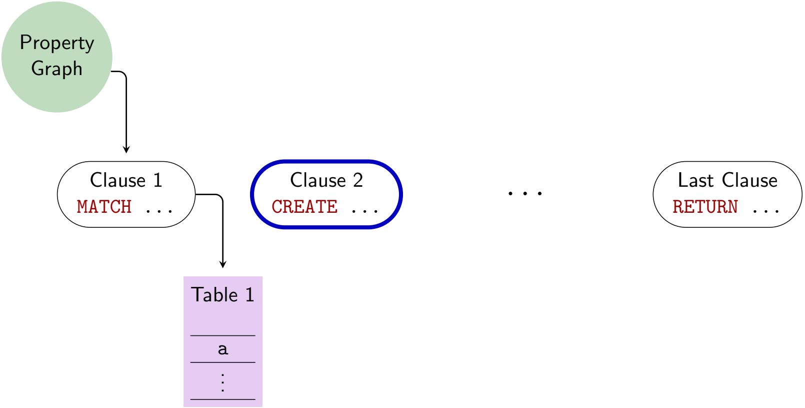
Part II: Neo4j, Property graphs and Cypher

6. Updating the property graph

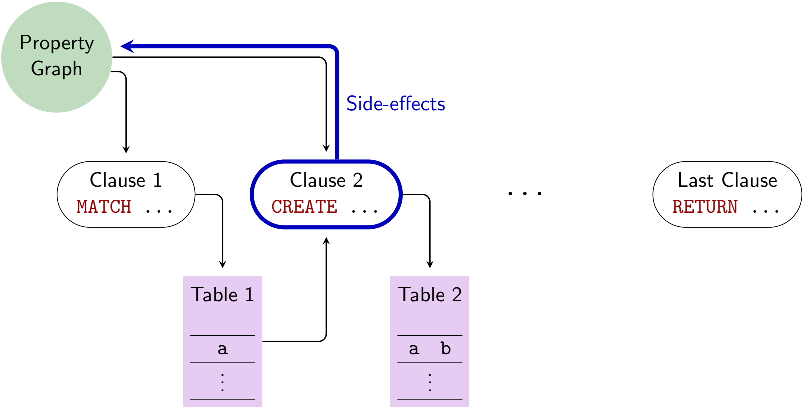
How evaluation works with update clauses



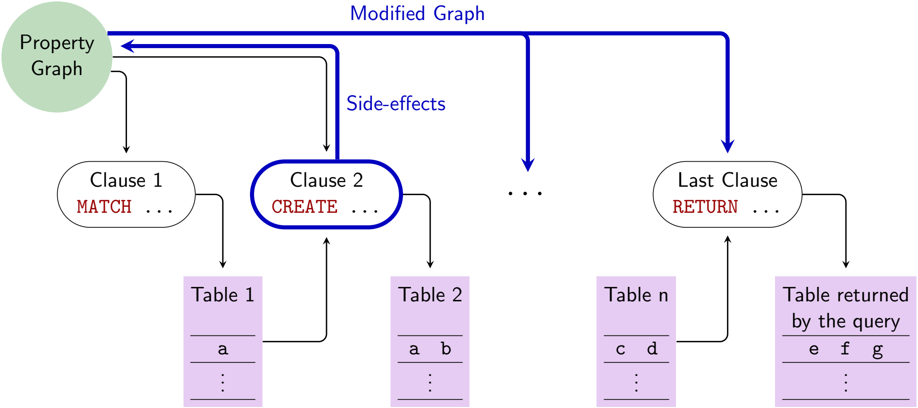
How evaluation works with update clauses



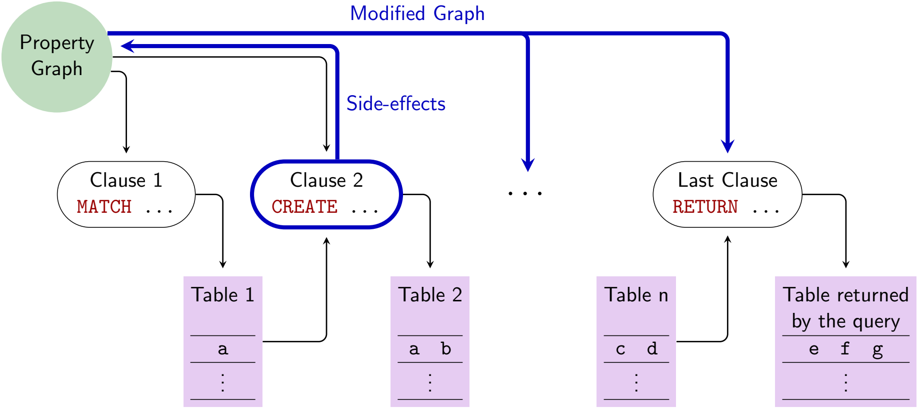
How evaluation works with update clauses



How evaluation works with update clauses



How evaluation works with update clauses



Neo4j complies to ACID

A $\Longrightarrow$ Modifications are **undone** if evaluation fails

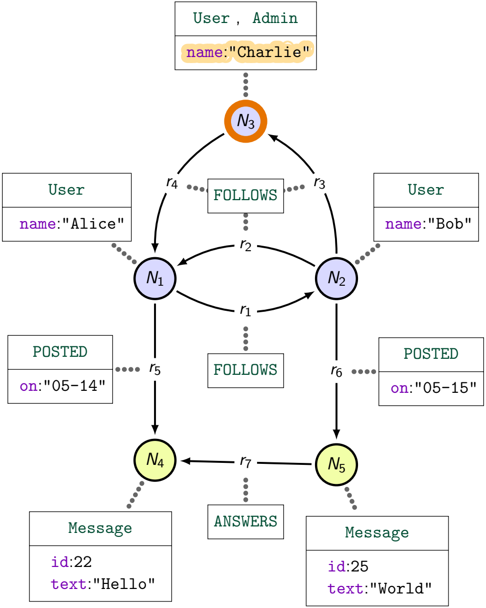
C $\Longrightarrow$ The PG must complies to IC **at the end** of evaluation only

I $\Longrightarrow$ Modifications are **invisible** to concurrent queries

Create nodes and relations (1)

- `CREATE (a:User {name:"Alice"})`
  - Creates a new node
  - Stores it in column `a`
- `CREATE (a)-[e:POSTED {on:"12-07"}]->(b)`
  - Creates a new relation from `a` to `b`
  - If `a` the input table has no column named `a`, creates a new node
  - Idem for `b`
  - Stores the new relation in column `e`

Create nodes and relations (2)



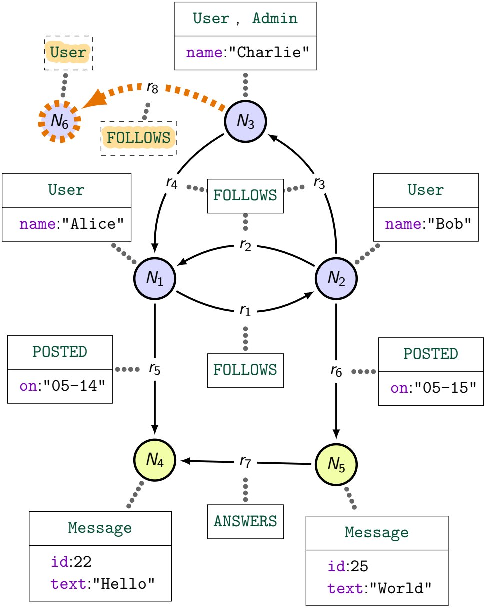
Query:

```
MATCH (a {name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                  (b:User)
```

Table after MATCH clause:

a
N_3

Create nodes and relations (2)

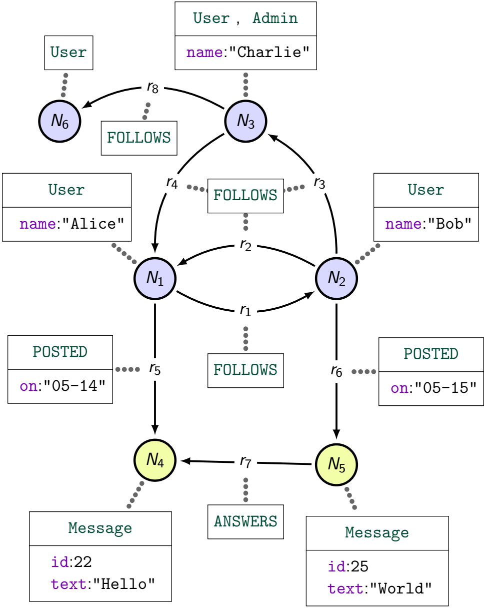


Query:

```
MATCH (a {name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                    (b:User)
```

Table after MATCH clause:

a
N_3



Query:

```
MATCH (a {name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                    (b:User)
```

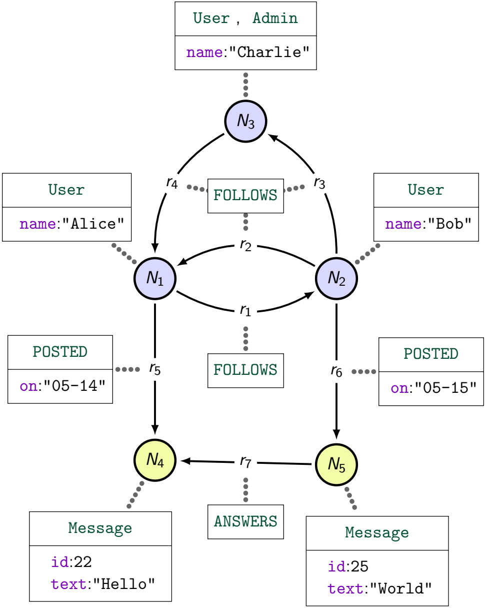
Table after MATCH clause:

a
N_3

Table after CREATE clause:

a	b
N_3	N_6

The example graph stored as CREATE clauses



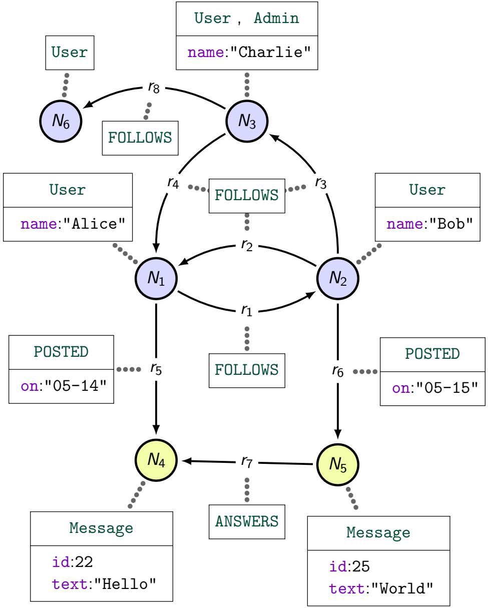
Query:

```
CREATE
  (n1:User{name:"Alice"}),
  (n2:User{name:"Bob"}),
  (n3:User:Admin
             {name:"Charlie"}),
  (n4:Message {id:22,
                text:"Hello"}),
  (n5:Message {id:25,
                text:"World"})
CREATE
  (n1)-[:FOLLOWS]->(n2),
  (n1)-[:POSTED
           {on:"05-04"}]->(n4),
  (n2)-[:FOLLOWS]->(n1),
  (n2)-[:FOLLOWS]->(n3),
  (n2)-[:POSTED
           {on:"05-04"}]->(n5),
  (n3)-[:FOLLOWS]->(n1),
  (n5)-[:ANSWERS]->(n4),
```

Delete nodes and relations

- `DELETE a`
  - If column a contains relations, delete them
  - If column a contains node:
    - if none of them has adjacent relation, delete them
    - otherwise the query fails.
- `DETACH DELETE a`
  - If column a contains relations, delete them
  - If column a contains nodes, delete them as well as every adjacent relations.

Modifying labels and properties (1)



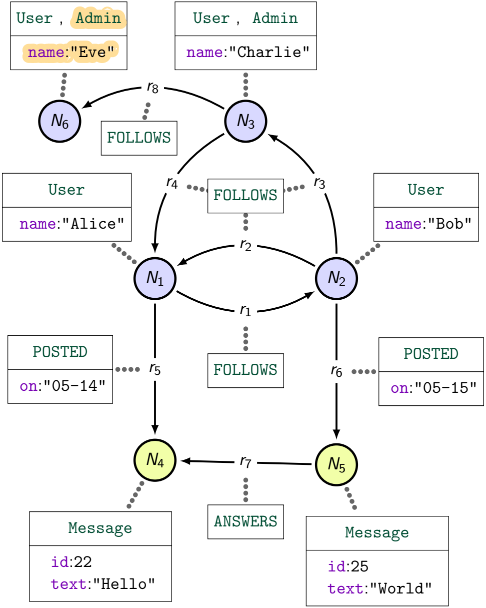
Query:

```
MATCH (a{name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                        (b:User)
SET b:Admin, b.name="Eve"
```

Table after CREATE clause:

a	b
N_3	N_6

Modifying labels and properties (1)

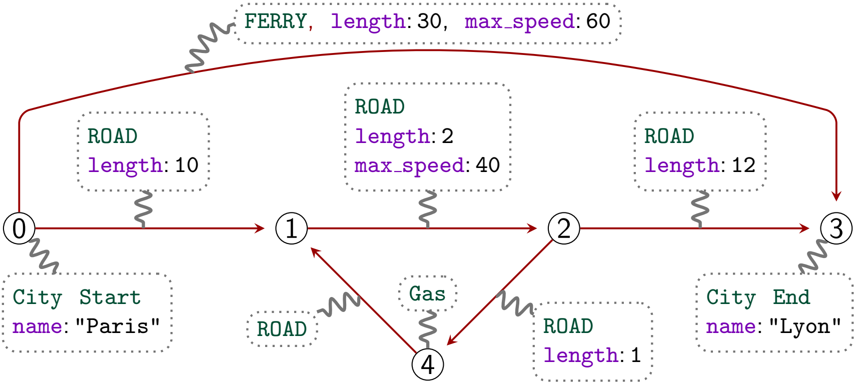


Query:

```
MATCH (a{name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                    (b:User)
SET b:Admin, b.name="Eve"
```

Table after CREATE clause:

a	b
N_3	N_6




```
MATCH ()-[e:ROAD]->()
    WHERE e.max_speed IS NULL
SET e.max_speed=80
```

⟹ Adds the property max_speed:80 to all ROAD that do not have one.

Appendix

Navigable outline I

Part II: Neo4j, Property graphs and Cypher

English-French translation I

English	French
Acyclic	Acyclique, Acircuitique
Bag, multiset	Multi-ensemble
Data model (DM)	Modèle de données
Edge	Arête, Arc
Endpoints	Extrémités
Key	Clef
Label	Etiquette
Match	
Node	Noeud
Path	Chemin
Pattern matching	Recherche de motif
Property, Attribute	Propriété, Attribut

English-French translation II

Property Graph (PG)	Graphe à propriétés, Graphe de propriété, Graphe attribué
Regular Path Query (RPQ)	
Semantics	Semantique
Set	Ensemble
Source	Source
Target	Destination
Trail	
Type	Type
Value	Valeur
Vertex	Sommet
Walk	Chemin, Marche