Introduction to querying with RPQs

semantics in theory and practice

Victor MARSAULT

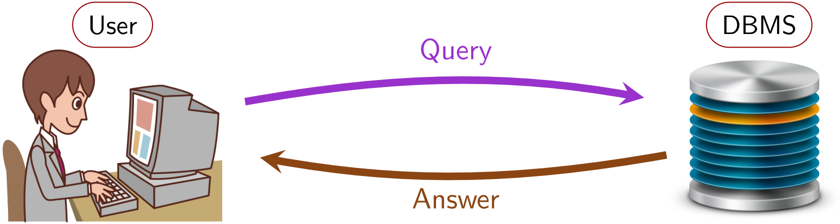
Université Gustave-Eiffel, CNRS, LIGM

Séminaire Automates, IRIF

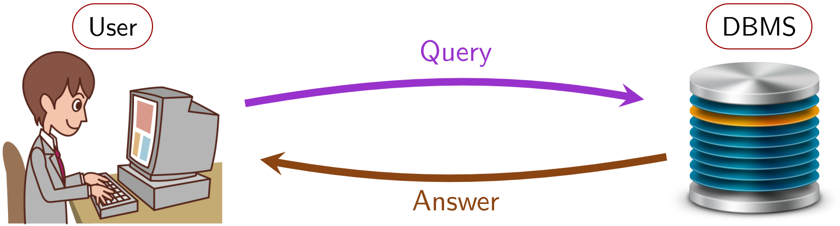
2023-09-22

Introduction

General setting

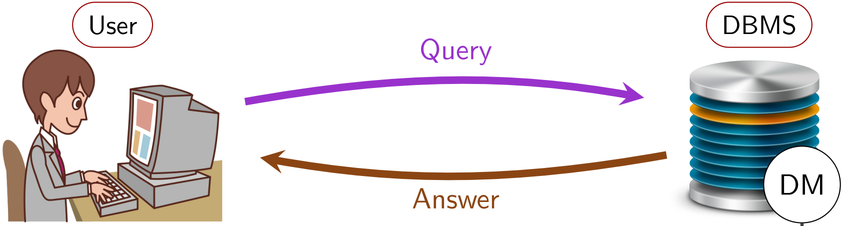


- DBMS = **D**ata**B**ase **M**anagement **S**ystem



General setting

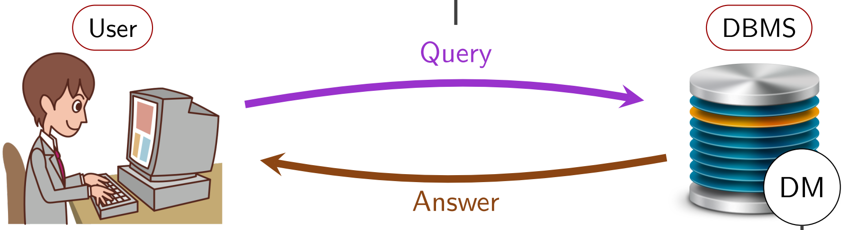
- DBMS = **D**ata**B**ase **M**anagement **S**ystem



- DM = **D**ata **M**odel = "*The way data is structured*"
  - Relational ? XML ? Property graph ? RDF ? etc.

General setting

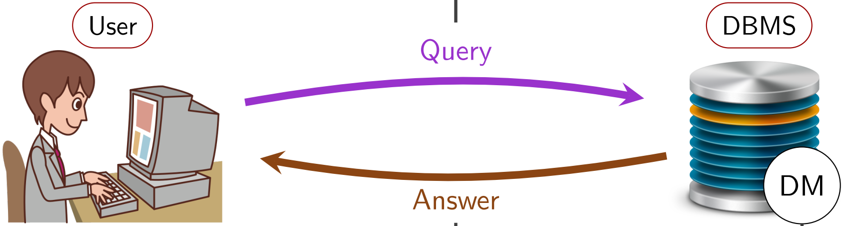
- DBMS = **D**ata**B**ase **M**anagement **S**ystem
- Query language
  - "*What can users ask for?*"



- DM = **D**ata **M**odel = "*The way data is structured*"
  - Relational ? XML ? Property graph ? RDF ? etc.

General setting

- DBMS = **D**ata**B**ase **M**anagement **S**ystem
- Query language
  - "*What can users ask for?*"



- Semantics of query
  - "*What does the query mean?*"
- DM = **D**ata **M**odel = "*The way data is structured*"
  - Relational ? XML ? Property graph ? RDF ? etc.

Vast majority of DMBS's are relational, not graph

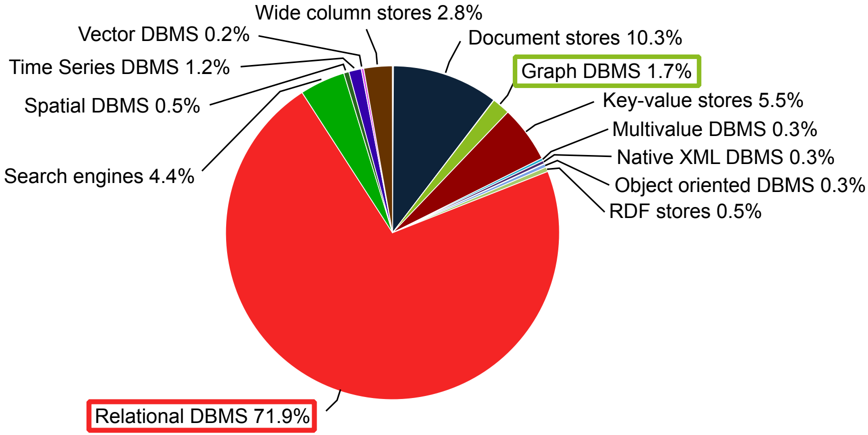


Figure and data from `db-engines.com`, August 2023

Graph DBMS is growing in popularity

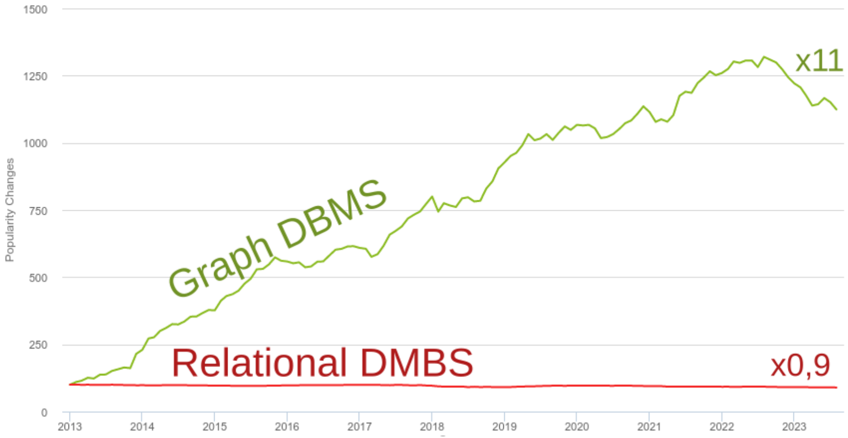


Figure and data from `db-engines.com`, August 2023

Relational DM = tables with cross-references

Example: DB for a small store

Client table

name	address
Alice	Wonderland
Bob	124 Conch St.
Charlie	1593 Broadway

Product table

name	price
Sponge	1€
Broom	5€
Rabbit	0€
Pocket Watch	100€

Relational DM = tables with cross-references

Example: DB for a small store

Client table

name	address
Alice	Wonderland
Bob	124 Conch St.
Charlie	1593 Broadway

Product table

name	price
Sponge	1€
Broom	5€
Rabbit	0€
Pocket Watch	100€

Order table

id	buyer	date
0	Alice	01-11-1865
1	Bob	07-07-2022

Order-content table

order_id	product
0	Rabbit
0	Pocket Watch
1	Sponge
1	Broom

Relational DM = tables with cross-references

Example: DB for a small store

Client table

name	address
Alice	Wonderland
Bob	124 Conch St.
Charlie	1593 Broadway

Product table

name	price
Sponge	1€
Broom	5€
Rabbit	0€
Pocket Watch	100€

Order table

id	buyer	date
0	Alice	01-11-1865
1	Bob	07-07-2022

Order-content table

order_id	product
0	Rabbit
0	Pocket Watch
1	Sponge
1	Broom

Relational DM = tables with cross-references

Example: DB for a small store

Client table

name	address
Alice	Wonderland
Bob	124 Conch St.
Charlie	1593 Broadway

Product table

name	price
Sponge	1€
Broom	5€
Rabbit	0€
Pocket Watch	100€

Order table

id	buyer	date
0	Alice	01-11-1865
1	Bob	07-07-2022

Order-content table

order_id	product
0	Rabbit
0	Pocket Watch
1	Sponge
1	Broom

Relational DM = tables with cross-references

Example: DB for a small store

Client table

name	address
Alice	Wonderland
Bob	124 Conch St.
Charlie	1593 Broadway

Product table

name	price
Sponge	1€
Broom	5€
Rabbit	0€
Pocket Watch	100€

Order table

id	buyer	date
0	Alice	01-11-1865
1	Bob	07-07-2022

Order-content table

order_id	product
0	Rabbit
0	Pocket Watch
1	Sponge
1	Broom

Relational DM = tables with cross-references

Example: DB for a small store

Client table

name	address
Alice	Wonderland
Bob	124 Conch St.
Charlie	1593 Broadway

Product table

name	price
Sponge	1€
Broom	5€
Rabbit	0€
Pocket Watch	100€

Order table

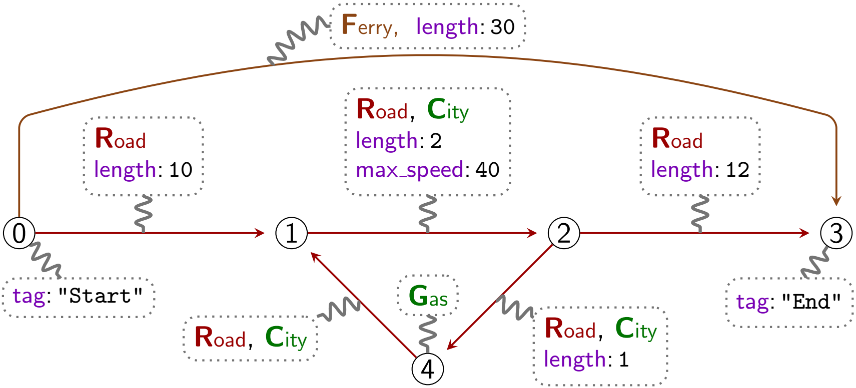
id	buyer	date
0	Alice	01-11-1865
1	Bob	07-07-2022

Order-content table

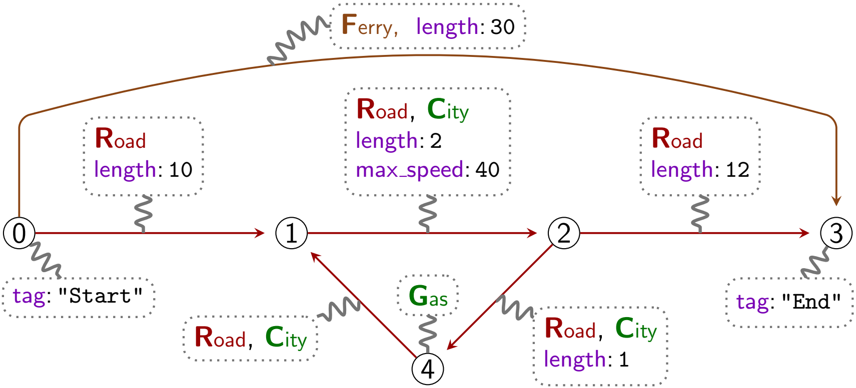
order_id	product
0	Rabbit
0	Pocket Watch
1	Sponge
1	Broom

Why use graphs ?

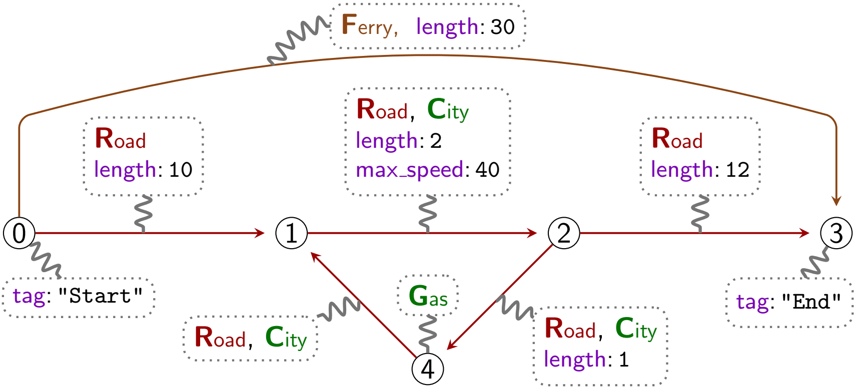
Some data have intrinsically the structure of graphs (e.g. networks)



Why not store graphs in tables?

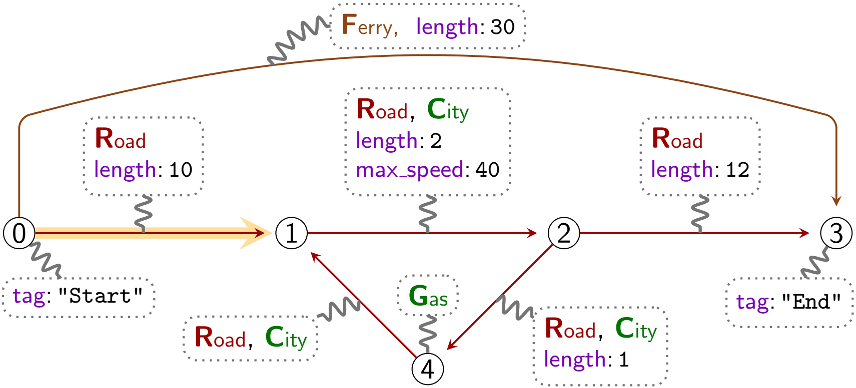


Why not store graphs in tables?



id	source_id	target_id	Road	Ferry	City	length	max_speed
e_{01}	0	1	true	false	false	10	
e_{12}	1	2	true	false	true	10	40
e_{24}	2	4	true	false	true		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

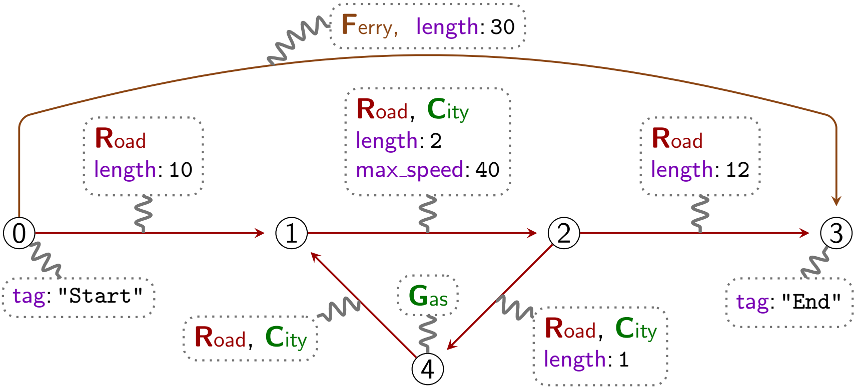
Why not store graphs in tables?



	id	source_id	target_id	Road	Ferry	City	length	max_speed	
▶	e_{01}	0	1	true	false	false	10		◀
	e_{12}	1	2	true	false	true	10	40	
	e_{24}	2	4	true	false	true			
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Why not store graphs in tables?

→ Model restriction allows navigational algorithms



id	source_id	target_id	Road	Ferry	City	length	max_speed
e_{01}	0	1	true	false	false	10	
e_{12}	1	2	true	false	true	10	40
e_{24}	2	4	true	false	true		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

History of query languages for property graphs

From RPQs to GQL: history and actors

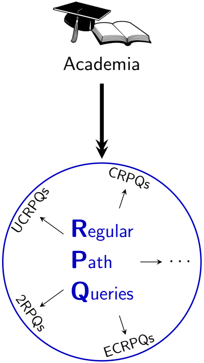
1987 – RPQs are invented [Cruz-Mendelzon-Wood 1987]

Academia

Regular
Path
Queries

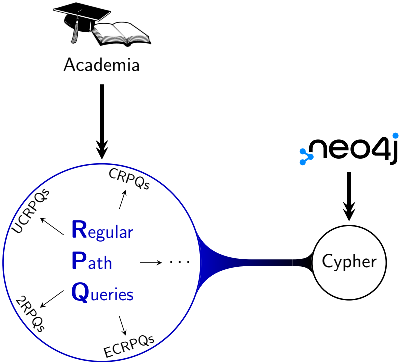
From RPQs to GQL: history and actors

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



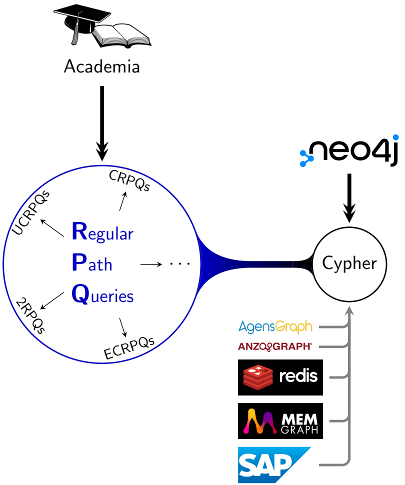
From RPQs to GQL: history and actors

2011 – Cypher is designed by Neo4j



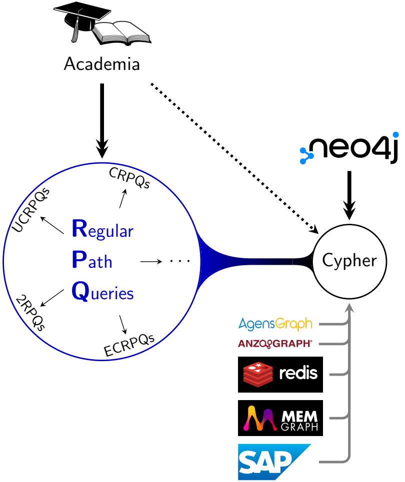
From RPQs to GQL: history and actors

mid 2010's – Cypher is successful and spreading. Standardize Cypher?



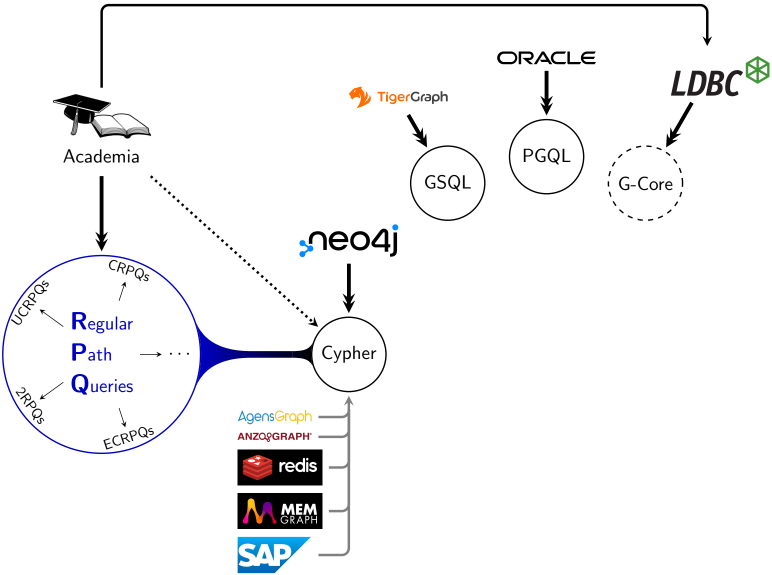
From RPQs to GQL: history and actors

mid 2010's – Cypher is successful and spreading. Standardize Cypher?



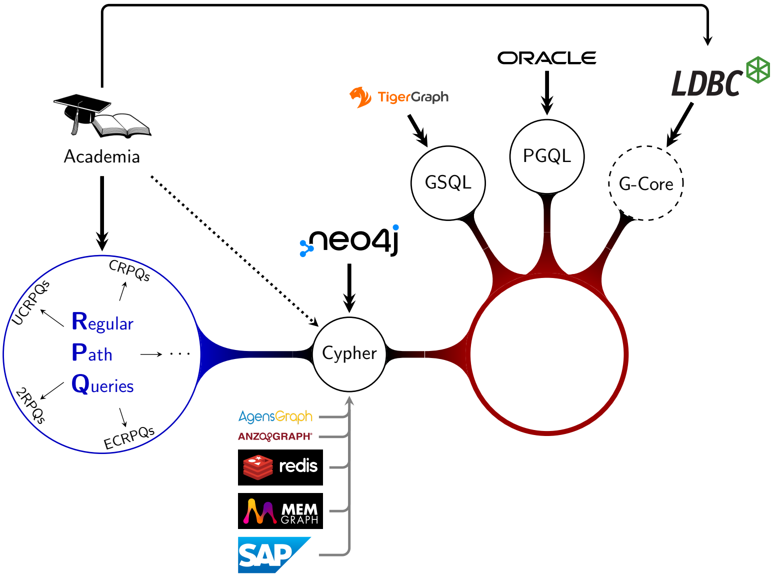
From RPQs to GQL: history and actors

mid 2010's – Other languages/DBMS are released



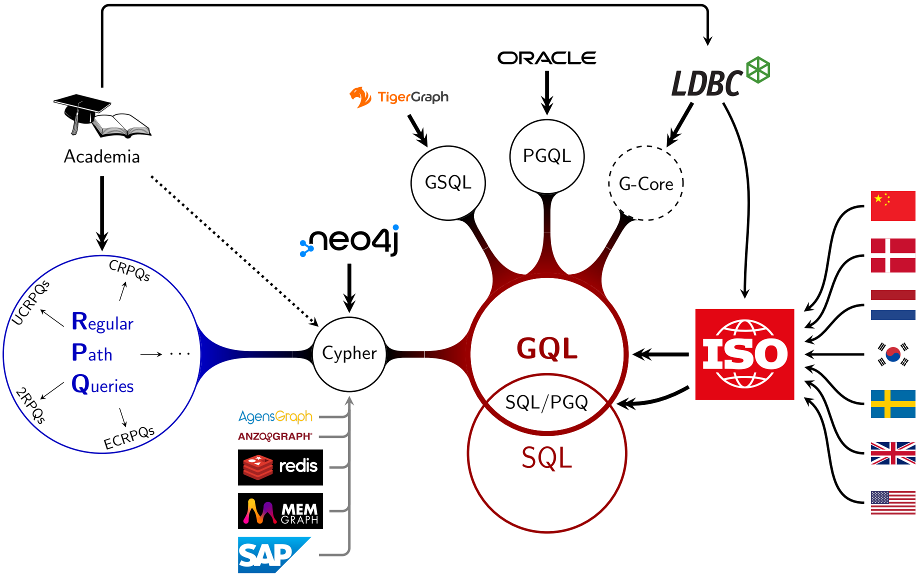
From RPQs to GQL: history and actors

late 2010's – Merge all existing languages ?



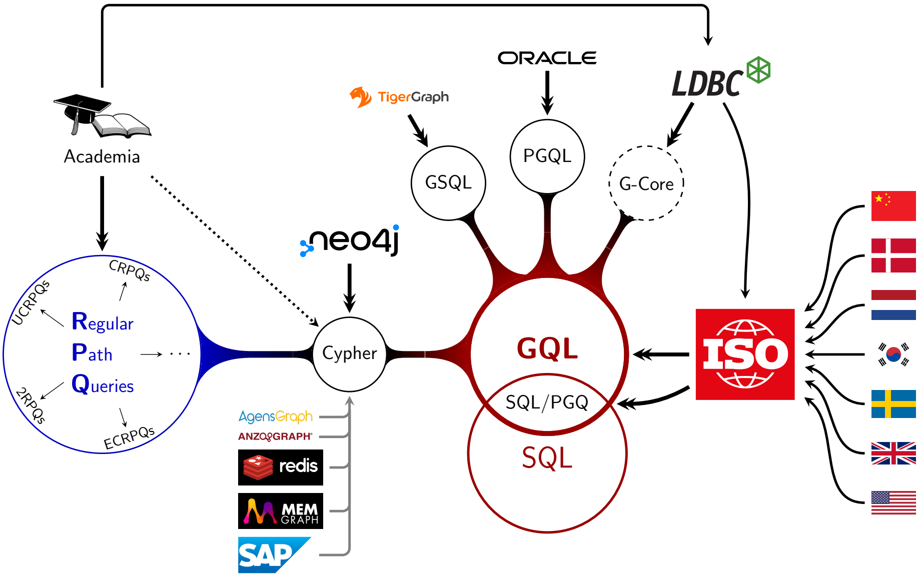
From RPQs to GQL: history and actors

2019-2021 – Two ISO projects: GQL and SQL/PGQ



From RPQs to GQL: history and actors

2024 (expected) – Publication of version 1 of GQL



Foundation of querying graph databases: RPQs

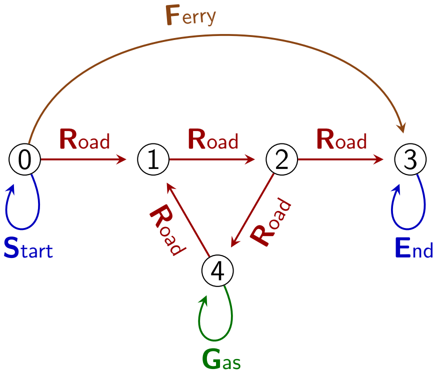
RPQs operates on labeled graphs

A graph consists of ...

- Vertices (or Nodes)
- Edges (or Relationships)
- Edge labels: {**R**, **F**, **G**, **S**, **E**}

Walk

- a.k.a. *path*
- Sequence of edges
- May reuse vertices and edges
- Is labeled by a word



RPQs operates on labeled graphs

A graph consists of ...

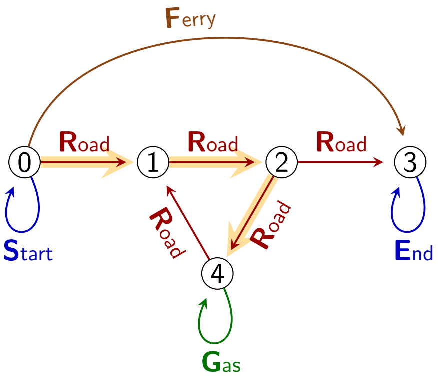
- Vertices (or Nodes)
- Edges (or Relationships)
- Edge labels: {**R**, **F**, **G**, **S**, **E**}

Walk

- a.k.a. *path*
- Sequence of edges
- May reuse vertices and edges
- Is labeled by a word

$0 \to 1 \to 2 \to 4$

is labeled by **RRR**



RPQs operates on labeled graphs

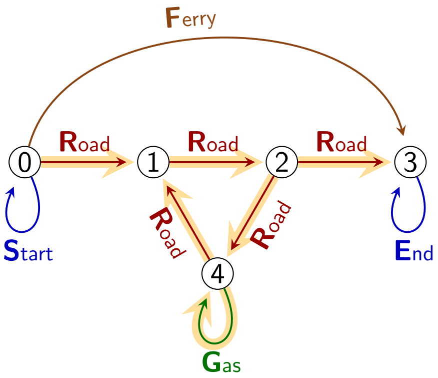
A graph consists of ...

- Vertices (or Nodes)
- Edges (or Relationships)
- Edge labels: {**R**, **F**, **G**, **S**, **E**}

Walk

- a.k.a. *path*
- Sequence of edges
- May reuse vertices and edges
- Is labeled by a word

$0 \to 1 \to 2 \to 4 \to 4 \to 1 \to 2 \to 3$ is labeled by **RRRGRRR**



RPQ = Regular expression

$$\mathcal{Q} ::= \begin{array}{l} \mathbf{A} \\ \mathcal{Q}\mathcal{Q} \\ \mathcal{Q} + \mathcal{Q} \\ \mathcal{Q}^* \end{array}$$

where **A** is a label in the graph.

RPQ = Regular expression

$$\mathcal{Q} ::= \begin{array}{l} \mathbf{A} \\ \mathcal{Q}\mathcal{Q} \\ \mathcal{Q} + \mathcal{Q} \\ \mathcal{Q}^* \end{array}$$

where **A** is a label in the graph.

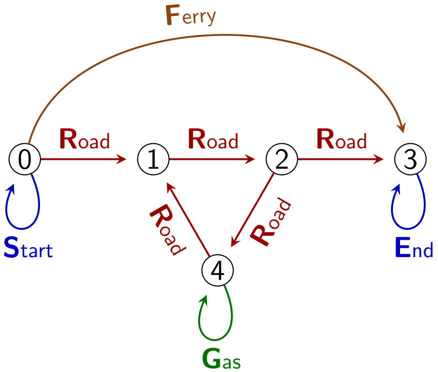
Matches

A match for $\mathcal{Q}$ is any walk w such that $\mathcal{Q}$ denotes the label of w

First example

Query RR matches...

...walks of two **R**oad-edges

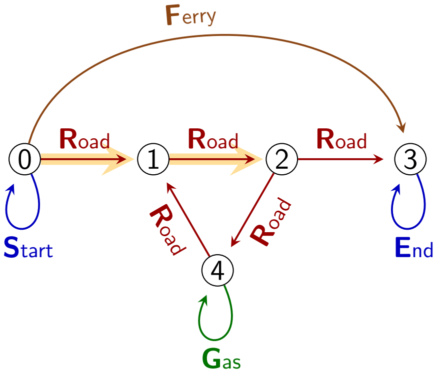


First example

Query RR matches...

...walks of two Road-edges

$0 \to 1 \to 2$	$2 \to 4 \to 1$
$1 \to 2 \to 3$	$4 \to 1 \to 2$
$1 \to 2 \to 3$	

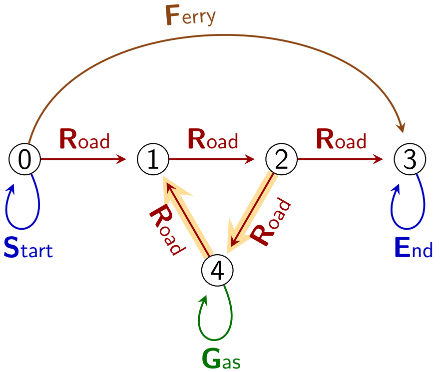


First example

Query **RR** matches...

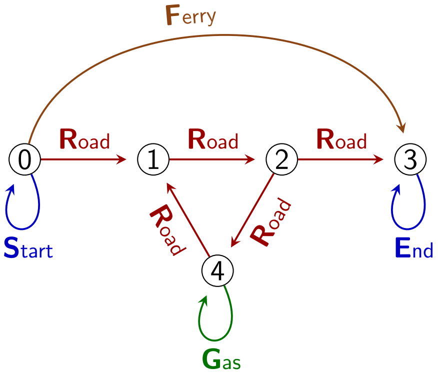
...walks of two **R**oad-edges

$0 \to 1 \to 2$	$2 \to 4 \to 1$
$1 \to 2 \to 3$	$4 \to 1 \to 2$
$1 \to 2 \to 3$	



$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

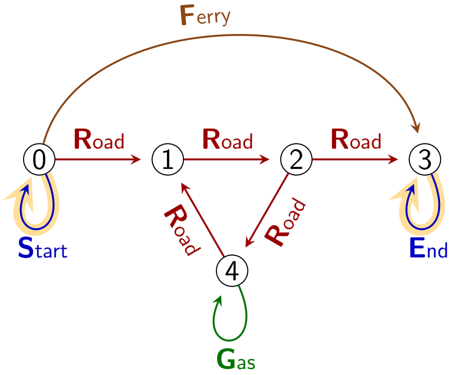
$\mathcal{Q}_1$ matches...



The queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$

$\mathcal{Q}_1 = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

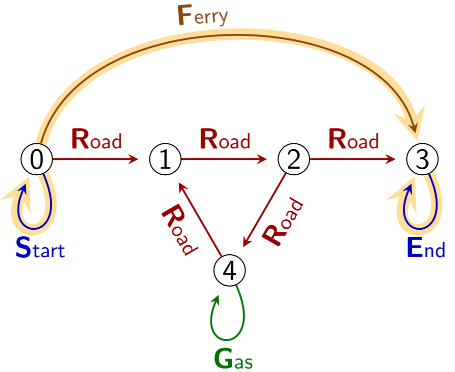
$\mathcal{Q}_1$ matches...



$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

$\mathcal{Q}_1$ matches...

- The ferry

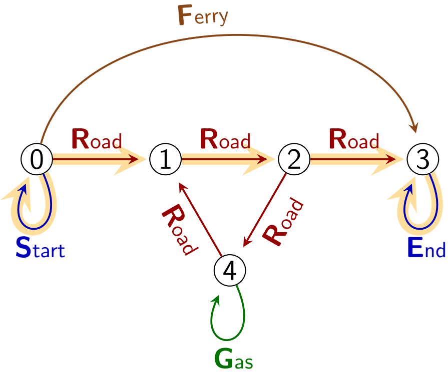


The queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$

$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

$\mathcal{Q}_1$ matches...

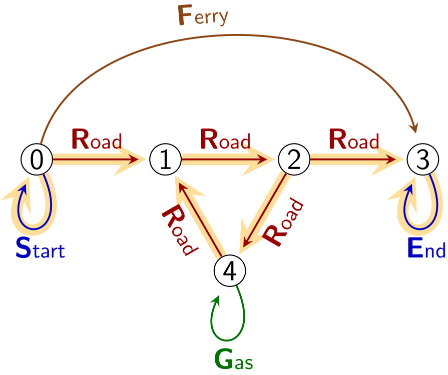
- The ferry
- The direct road



$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

$\mathcal{Q}_1$ matches...

- The ferry
- The direct road
- Roads with laps in the circuit



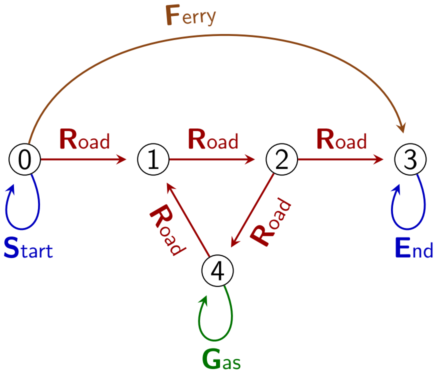
$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

$\mathcal{Q}_1$ matches...

- The ferry
- The direct road
- Roads with laps in the circuit

$\mathcal{Q}_2 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{G}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

$\mathcal{Q}_2$ matches...



The queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$

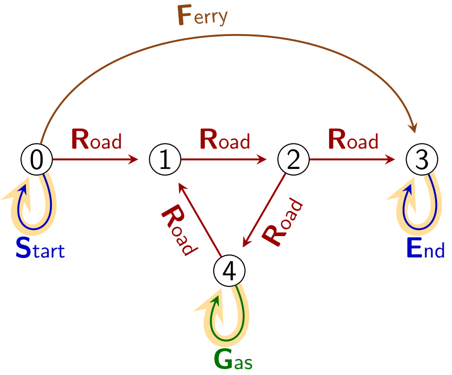
$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

$\mathcal{Q}_1$ matches...

- The ferry
- The direct road
- Roads with laps in the circuit

$\mathcal{Q}_2 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{G}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

$\mathcal{Q}_2$ matches...



The queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$

$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

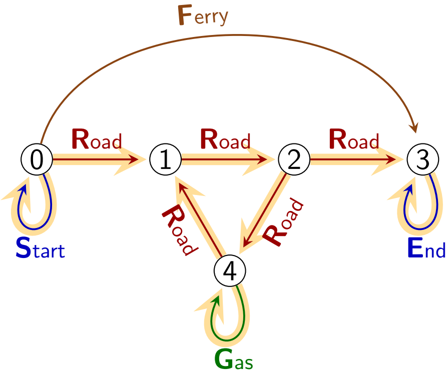
$\mathcal{Q}_1$ matches...

- The ferry
- The direct road
- Roads with laps in the circuit

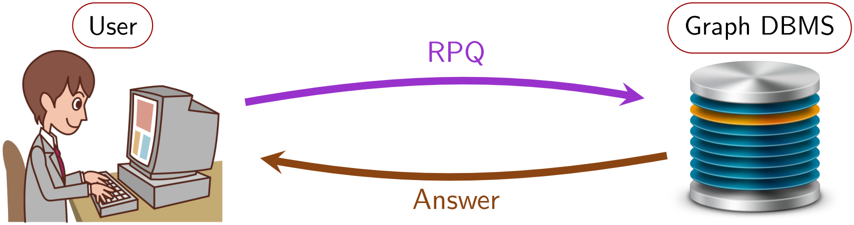
$\mathcal{Q}_2 = \mathbf{S}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{G}\,(\mathbf{R}+\mathbf{F})^*\,\mathbf{E}$

$\mathcal{Q}_2$ matches...

- Roads with laps in the circuit



Fundamental challenge with RPQs



⚠ Infinitely many matches but finite answer ⚠

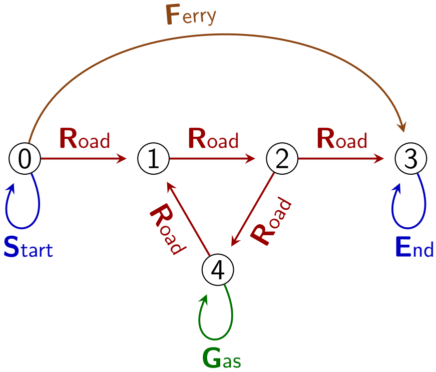
Semantics of RPQs

Homomorphism semantics

Main theoretical semantics [Angles et al. 2017], used in SparQL

Definition

- Returns the endpoints of matches



Main theoretical semantics [Angles et al. 2017], used in SparQL

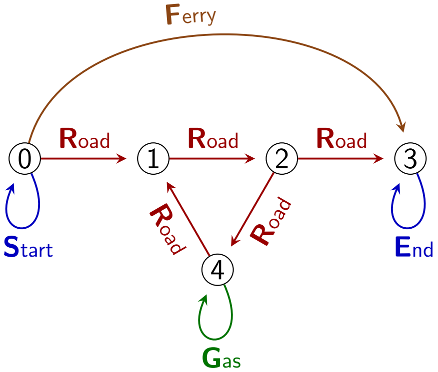
Definition

- Returns the endpoints of matches

$\mathcal{Q}_1 = \quad \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$

$\mathcal{Q}_2 = \quad \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$

- All matches are of the form: $0 \to \cdots \to 3$
$\Rightarrow \mathcal{Q}_1$ and $\mathcal{Q}_2$ return $\{(0,3)\}$



Homomorphism semantics (2)

Pros and cons

Pros

- Efficient algorithms
- Well grounded theory

Homomorphism semantics (2)

Pros and cons

Pros

- Efficient algorithms
- Well grounded theory

Cons

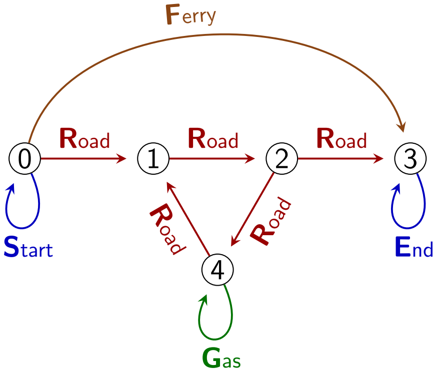
- Very limited information in the answer
  - User: *"I want to go from LIGM to IRIF by public transportation"*
  - Database: *"Yes you can"*

Shortest-walk semantics

Used in PGQL (Oracle), GSQL (TigerGraph) and G-core [Angles et al. 2018]

Definition

- Return the walk with the least number of edges



Shortest-walk semantics

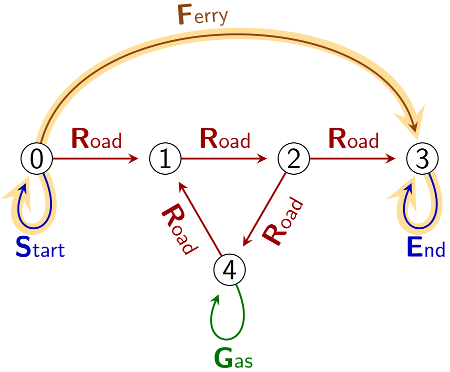
Used in PGQL (Oracle), GSQL (TigerGraph) and G-core [Angles et al. 2018]

Definition

- Return the walk with the least number of edges

$\mathcal{Q}_1 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$

- $\mathcal{Q}_1$ returns 1 walk
  - the ferry
- Walks taking the road have more edges



Shortest-walk semantics

Used in PGQL (Oracle), GSQL (TigerGraph) and G-core [Angles et al. 2018]

Definition

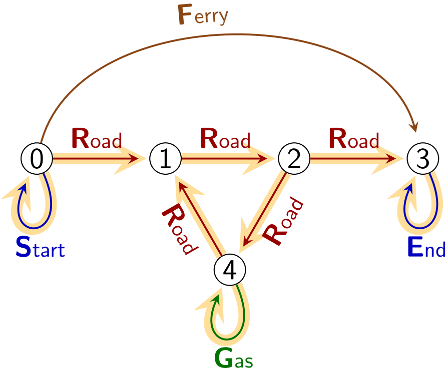
- Return the walk with the least number of edges

$\mathcal{Q}_1 = \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$

- $\mathcal{Q}_1$ returns 1 walk
  - the ferry
- Walks taking the road have more edges

$\mathcal{Q}_2 = \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$

- $\mathcal{Q}_1$ returns 1 walk
  - the one-lap road

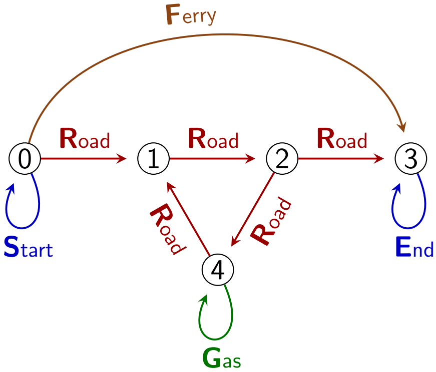


Trail semantics

Used in Cypher (Neo4j) [Francis et al. 2018]

Definition

- Return walks
- Forbid to repeat edges



Trail semantics

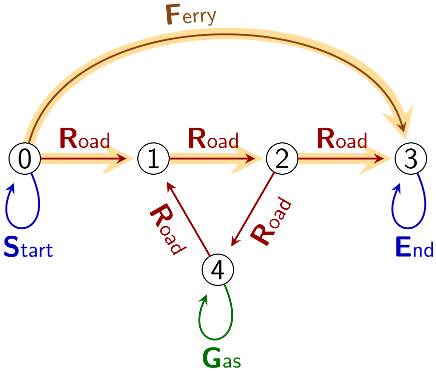
Used in Cypher (Neo4j) [Francis et al. 2018]

Definition

- Return walks
- Forbid to repeat edges

$$\mathcal{Q}_1 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- $\mathcal{Q}_1$ returns 2 walks
  - the ferry
  - the straight road



Trail semantics

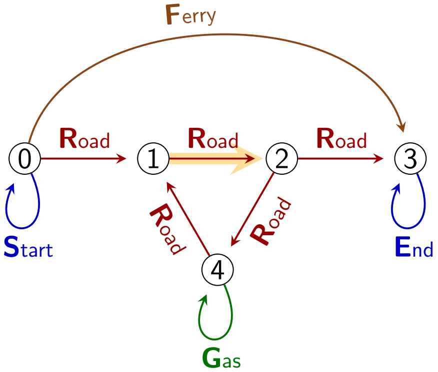
Used in Cypher (Neo4j) [Francis et al. 2018]

Definition

- Return walks
- Forbid to repeat edges

$$\mathcal{Q}_1 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- $\mathcal{Q}_1$ returns 2 walks
  - the ferry
  - the straight road
- Walks with circuit laps repeat the middle edge



Used in Cypher (Neo4j) [Francis et al. 2018]

Definition

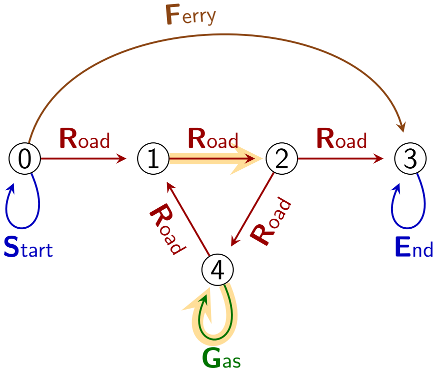
- Return walks
- Forbid to repeat edges

$$\mathcal{Q}_1 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- $\mathcal{Q}_1$ returns 2 walks
  - the ferry
  - the straight road
- Walks with circuit laps repeat the middle edge

$$\mathcal{Q}_2 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- $\mathcal{Q}_2$ returns nothing



Comparing trail and shortest-walk semantics

	Shortest-walk	Trail
Existence	▪ Tract.	▪ Untract.
Enumeration	▪ Tract.	▪ Untract.
Distinct Enum	▪ Tract.	▪ Untract.
Membership	▪ Tract.	▪ Tract.

Trail and shortest-walk have different goals

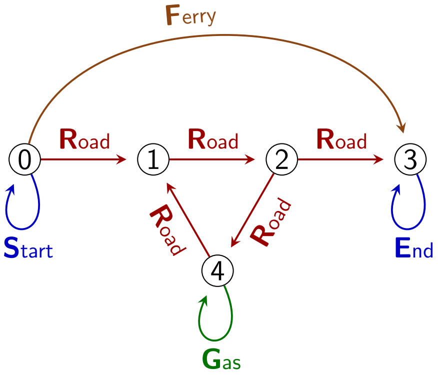
$$\mathcal{Q}_1 = \quad \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

Shortest-walk semantics

- outputs the **F**erry-walk only
- computes the "best" answer

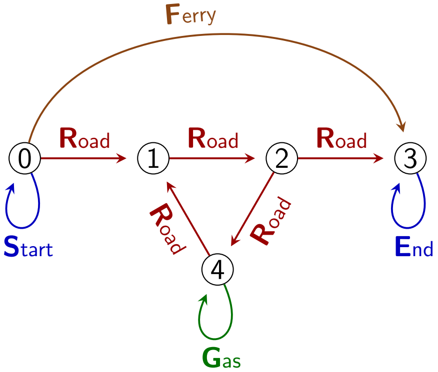
Trail semantics:

- outputs the **F**erry-walk and the direct **R**oad.
- computes "nonstupid" answers



"Coverage"

Possibilities in the match space that are in the output



"Coverage"

Possibilities in the match space that are in the output

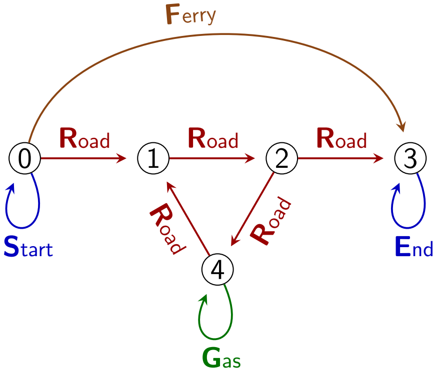
⇓ *Enables* ⇓

Vertical postprocessing

Compute something across returned walks.

Example:

- counting walk (connectivity)
- best walk w.r.t. some metrics



“Coverage”

Possibilities in the match space that are in the output

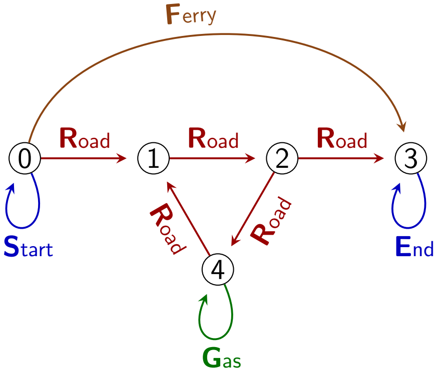
⇓ *Enables* ⇓

Vertical postprocessing

Compute something across returned walks.

Example:

- counting walk (connectivity)
- best walk w.r.t. some metrics



One little problem

- Define “coverage”

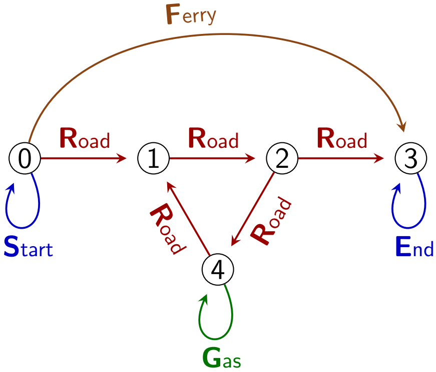
Trail semantics provide "bad" coverage

Fact

Trail sometimes discard walks that seem "nonstupid"

$$\mathcal{Q}_2 = \ \textbf{S}\ (\textbf{R}+\textbf{F})^*\ \textbf{G}\ (\textbf{R}+\textbf{F})^*\ \textbf{E}$$

- Trail outputs nothing



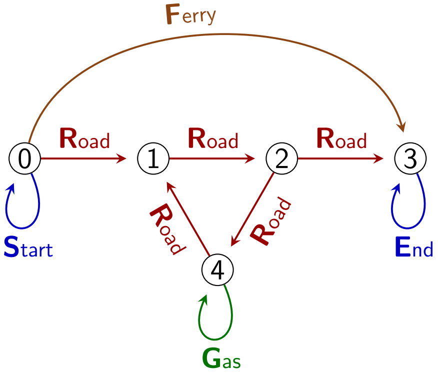
Trail semantics provide "bad" coverage

Fact

Trail sometimes discard walks that seem "nonstupid"

$$\mathcal{Q}_2 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- Trail outputs nothing



Two little problems

- Define "coverage"
- Define "good" coverage

Comparing trail and shortest-walk semantics

	Shortest-walk	Trail
Existence	■ Tract.	■ Untract.
Enumeration	■ Tract.	■ Untract.
Distinct Enum	■ Tract.	■ Untract.
Membership	■ Tract.	■ Tract.
Counting	■ Meaningless	■ Untract.
Coverage	■ None	■ Some, with no guarantee

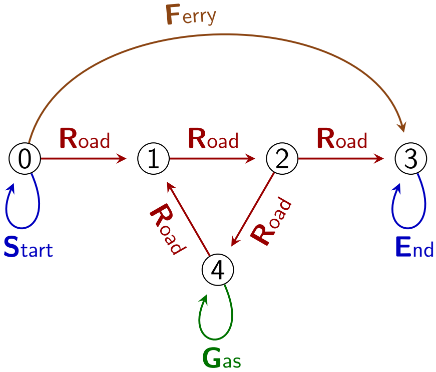
Run-based semantics

Binding-trail semantics (1)

[David-Francis-Marsault 2023]

Definition

- Returns walks
- Each edge may use each atom of Q at most once



Binding-trail semantics (1)

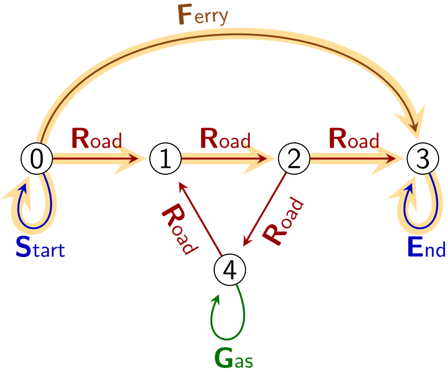
[David-Francis-Marsault 2023]

Definition

- Returns walks
- Each edge may use each atom of Q at most once

$$\mathcal{Q}_1 = \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- Returns
  - the ferry
  - the straight road



Binding-trail semantics (1)

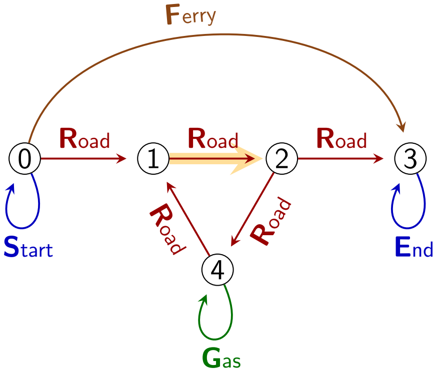
[David-Francis-Marsault 2023]

Definition

- Returns walks
- Each edge may use each atom of Q at most once

$$\mathcal{Q}_1 = \quad \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- Returns
  - the ferry
  - the straight road
- In walks with circuit laps
  $\implies$ the middle edge reuses **R**

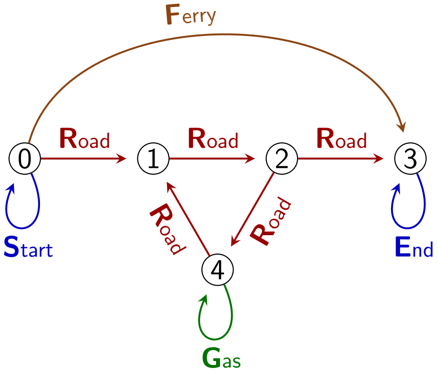


Binding-trail semantics (2)

[David-Francis-Marsault 2023]

Definition

- Returns walks
- Each edge may use each atom of Q at most once



Binding-trail semantics (2)

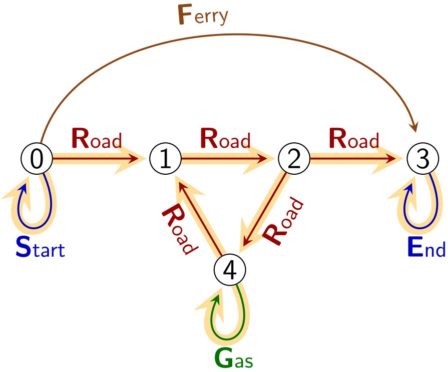
[David-Francis-Marsault 2023]

Definition

- Returns walks
- Each edge may use each atom of Q at most once

$$\mathcal{Q}_2 = \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- Returns the walk with one circuit lap



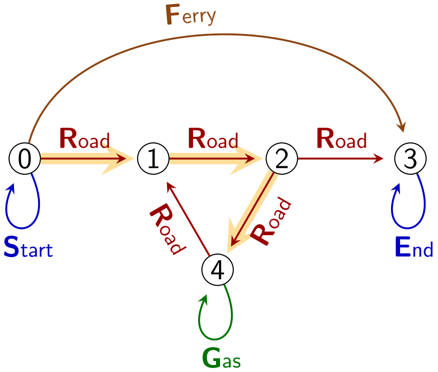
[David-Francis-Marsault 2023]

Definition

- Returns walks
- Each edge may use each atom of Q at most once

$$\mathcal{Q}_2 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

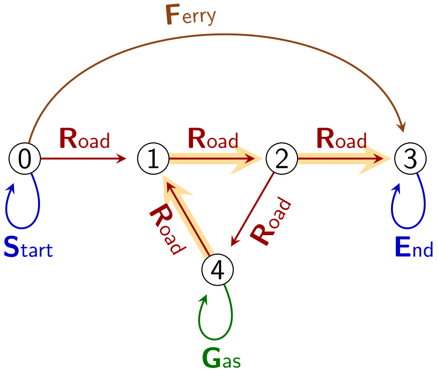
- Returns the walk with one circuit lap
  - Before **G** $\rightarrow$ use the left **R**



[David-Francis-Marsault 2023]

Definition

- Returns walks
- Each edge may use each atom of Q at most once



$$\mathcal{Q}_2 = \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- Returns the walk with one circuit lap
  - Before **G** $\rightarrow$ use the left **R**
  - After **G** $\rightarrow$ use the right **R**

Binding-trail semantics (2)

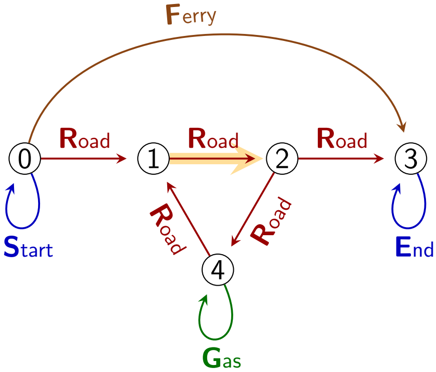
[David-Francis-Marsault 2023]

Definition

- Returns walks
- Each edge may use each atom of Q at most once

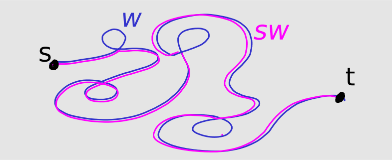
$$\mathcal{Q}_2 = \quad \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- Returns the walk with one circuit lap
  - Before **G** $\rightarrow$ use the left **R**
  - After **G** $\rightarrow$ use the right **R**
- In walks with 2+ circuit laps $\implies$ the middle edge reuses the left **R** or the right **R**



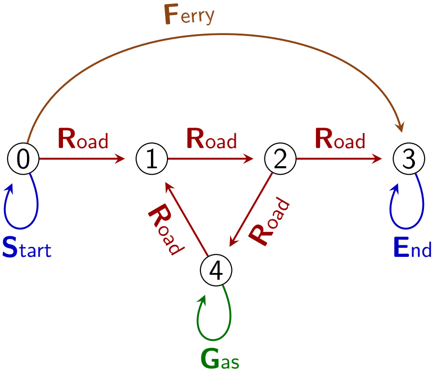
Binding trail provides some coverage guarantee

Lemma



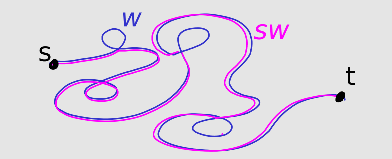
$\forall$ match w of Q

$\implies$ some subwalk sw returned



Binding trail provides some coverage guarantee

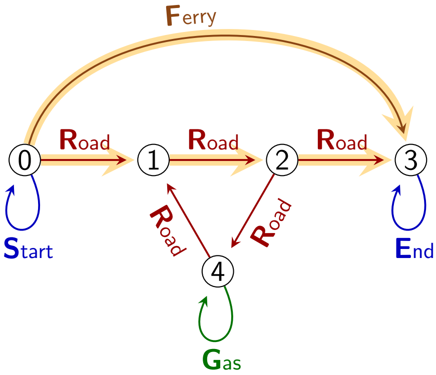
Lemma



$\forall$ match w of Q
$\implies$ some subwalk sw returned

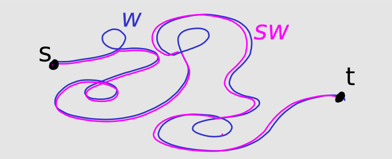
$$\mathcal{Q}_1 = \quad \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- The **F**erry-walk is returned
- The straight road is returned



Binding trail provides some coverage guarantee

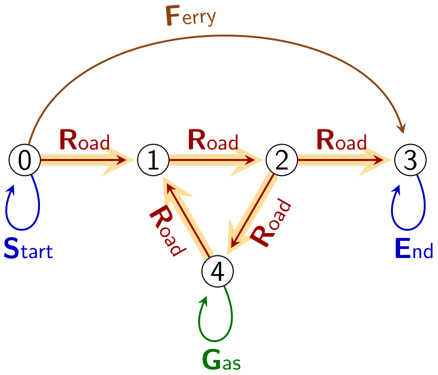
Lemma



$\forall$ match w of Q
$\implies$ some subwalk sw returned

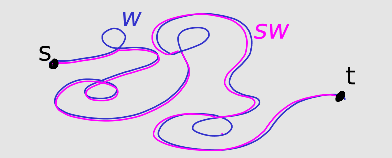
$$\mathcal{Q}_1 = \quad \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$$

- The **F**erry-walk is returned
- The straight road is returned
- Walks with $\geq$1 laps are "covered" by the later



Binding trail provides some coverage guarantee

Lemma

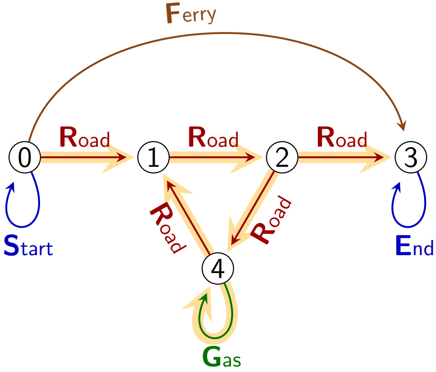


$\forall$ match w of Q

$\implies$ some subwalk sw returned

$\mathcal{Q}_1 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$

- The **F**erry-walk is returned
- The straight road is returned
- Walks with $\geq$1 laps are "covered" by the later



$\mathcal{Q}_2 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{F})^*\ \mathbf{E}$

- The 1-lap walk is returned
- Walks with $\geq$2 laps are "covered" by the later

Binding-trail is syntax-dependent

The output depends on the syntax of the query

$\mathbf{R}^*$

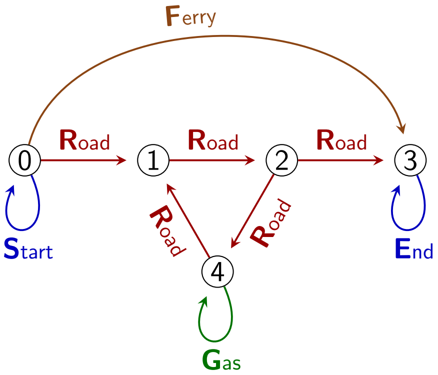
- allows no lap in the circuit

$\mathbf{R}^*\mathbf{R}^*$

- allows 1 lap in the circuit

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

- allows 1 lap in the circuit
- In general, $\neq \mathbf{R}^*\mathbf{R}^*$



Binding-trail is syntax-dependent

The output depends on the syntax of the query

$\mathbf{R}^*$

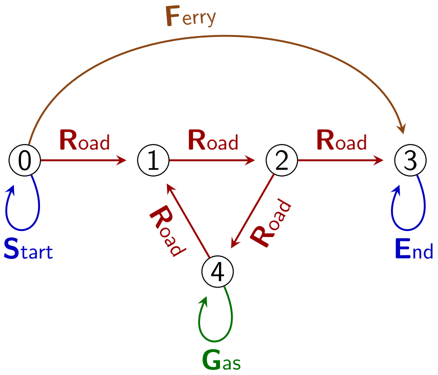
- allows no lap in the circuit

$\mathbf{R}^*\mathbf{R}^*$

- allows 1 lap in the circuit

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

- allows 1 lap in the circuit
- In general, $\neq \mathbf{R}^*\mathbf{R}^*$



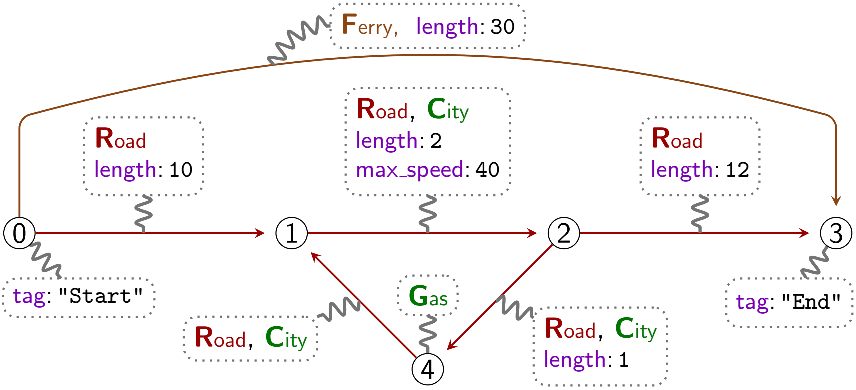
- Unusual from theoretical point of view
- The user has finer control on the output
- This kind of syntax quirks exists in practice

Comparing binding-trail to prior semantics

	Shortest-walk	Trail	Run-based
Existence	■ Tract.	■ Untract.	■ Tract.
Enumeration	■ Tract.	■ Untract.	■ Tract.
Distinct Enum	■ Tract.	■ Untract.	Open
Membership	■ Tract.	■ Tract.	■ Untract.
Counting	■ Meaningless	■ Untract.	■ Untract.
Coverage	■ None	■ Some, with no guarantee	■ Some, with some guarantee
Other			■ Syntax-depend.

Property graphs and real query languages

Back to our example property graph



Vertices and edges may bear:

- zero or more labels
- zero or more properties

- Property = key-value pair
- Key = string
- Value = `bool`, `int`, `str`, …

Cypher features

Cypher features

- Trail semantics
- Restricted RPQs (in fact UC2RPQs) with the following restrictions:
  - Under a Kleene star, only unions of atoms are allowed
- ASCII-art syntax
- Cypher is graph-to-tables
- Chaining of clauses

ASCII-art syntax

- Vertices: `MATCH (:Gas)`

- Vertices: `MATCH (:Gas)` `MATCH ({tag:"Start"})`

ASCII-art syntax

- Vertices: `MATCH (:Gas)` `MATCH ({tag:"Start"})`
- Edges: `MATCH -[:Road]->`

ASCII-art syntax

- Vertices: `MATCH (:Gas)` `MATCH ({tag:"Start"})`
- Edges: `MATCH -[:Road]->`

- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`

ASCII-art syntax

- Vertices: `MATCH (:Gas)` `MATCH ({tag:"Start"})`
- Edges: `MATCH -[:Road]->`

- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`
- Disjunction: `MATCH ()-[:Road|Ferry]->()`

ASCII-art syntax

- Vertices: `MATCH (:Gas)` `MATCH ({tag:"Start"})`
- Edges: `MATCH -[:Road]->`

- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`
- Disjunction: `MATCH ()-[:Road|Ferry]->()`
- Kleene star: `MATCH ()-[:Road*]->()`

ASCII-art syntax

- Vertices: `MATCH (:Gas)` `MATCH ({tag:"Start"})`
- Edges: `MATCH -[:Road]->`

- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`
- Disjunction: `MATCH ()-[:Road|Ferry]->()`
- Kleene star: `MATCH ()-[:Road*]->()`

- Variables: `MATCH ()-[:Road]->(x)-[:Road]->()`

ASCII-art syntax

- Vertices: `MATCH (:Gas)` `MATCH ({tag:"Start"})`
- Edges: `MATCH -[:Road]->`

- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`
- Disjunction: `MATCH ()-[:Road|Ferry]->()`
- Kleene star: `MATCH ()-[:Road*]->()`

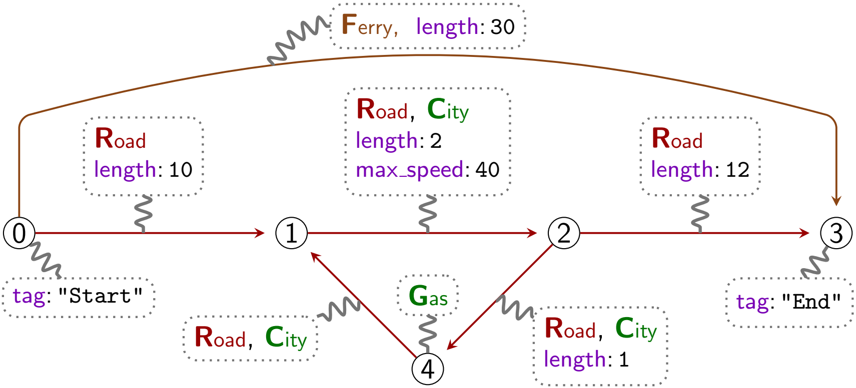
- Variables: `MATCH ()-[:Road]->(x)-[:Road]->()`

Cypher queries for $\mathcal{Q}_1$ and $\mathcal{Q}_2$

```
MATCH ({tag:"Start"})-[:Road|Ferry*]->({tag:"End"})
```

```
MATCH ({tag:"Start"})-[:Road|Ferry*]->
                    (:Gas)-[:Road|Ferry*]->({tag:"End"})
```

Cypher returns a table...



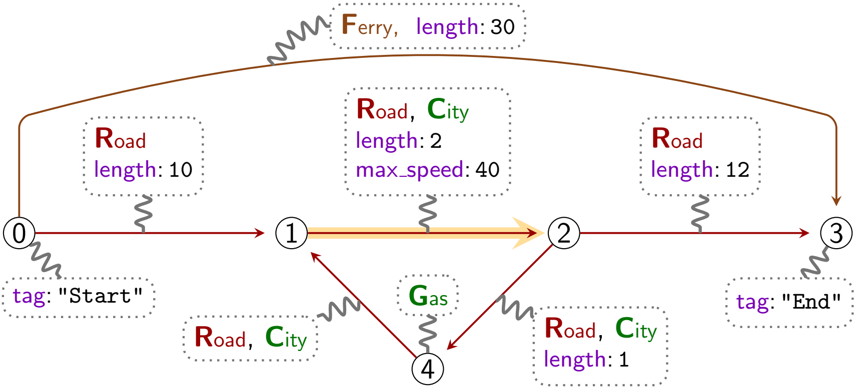
Query

```
MATCH (s)-[:City]->(t)
```

Result

s	t
1	2
2	4
4	1

Cypher returns a table...



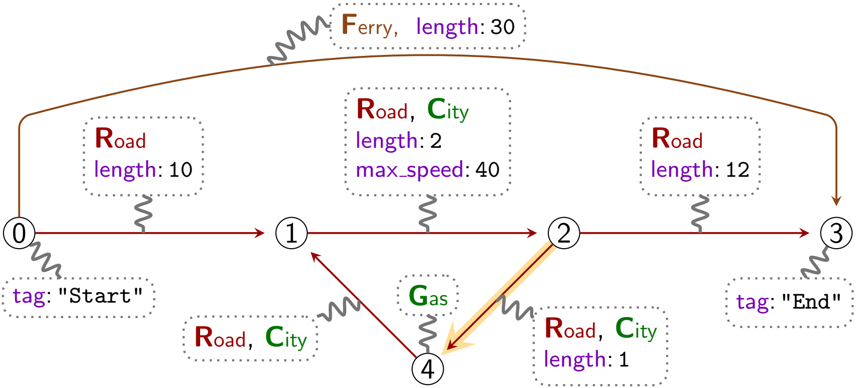
Query

```
MATCH (s)-[:City]->(t)
```

Result

s	t
1	2
2	4
4	1

Cypher returns a table...



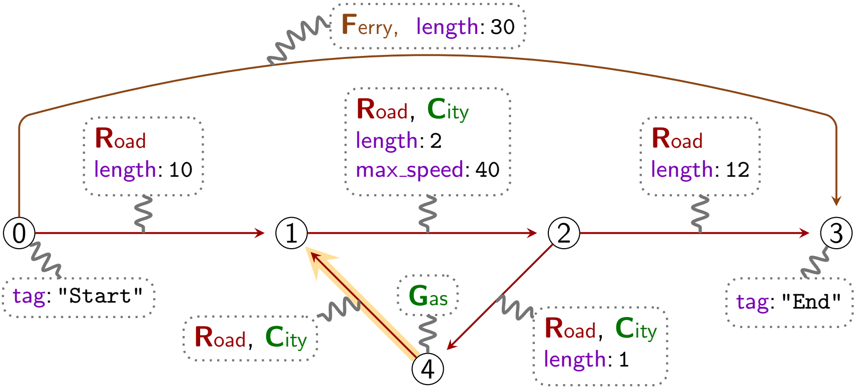
Query

```
MATCH (s)-[:City]->(t)
```

Result

s	t
1	2
2	4
4	1

Cypher returns a table...



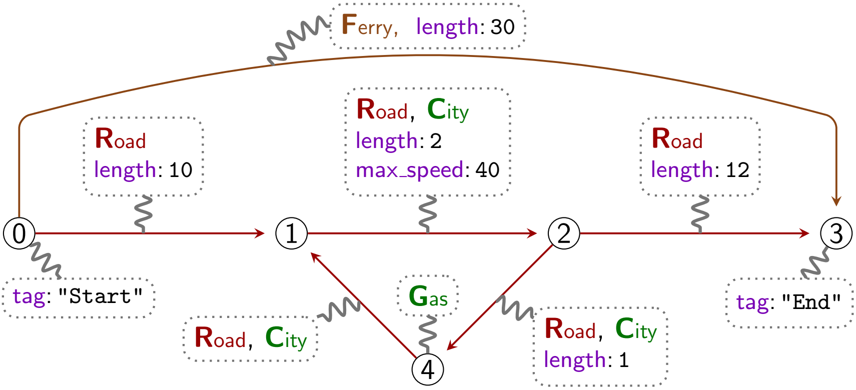
Query

```
MATCH (s)-[:City]->(t)
```

Result

s	t
1	2
2	4
4	1

Cypher returns a table... but computes walks



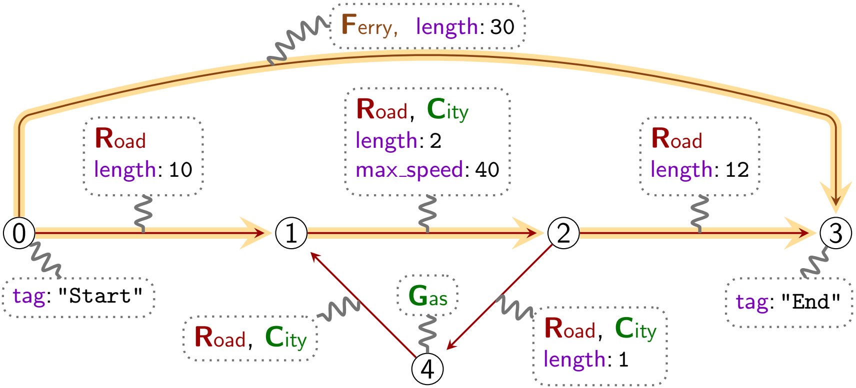
Query $\mathcal{Q}_1$

```
MATCH
    (s {tag:"Start"})
        -[:Road|Ferry*]->
            (t {tag:"End"})
```

Result

s	t
0	3
0	3

Cypher returns a table... but computes walks



Query $\mathcal{Q}_1$

```
MATCH
    (s {tag:"Start"})
        -[:Road|Ferry*]->
            (t {tag:"End"})
```

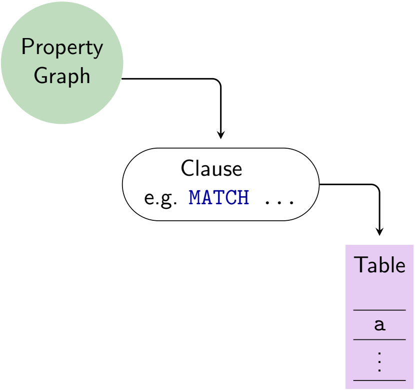
Result

s	t	
0	3	← The ferry
0	3	← The direct road

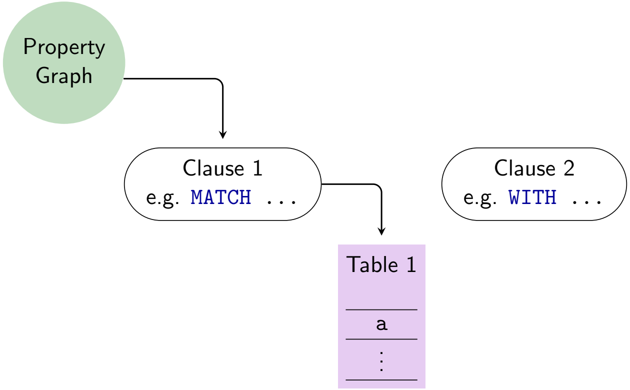
Other Cypher constructs

- `WHERE`: filter rows
- `WITH` or `RETURN`:
  - add/rename columns
  - horizontal aggregation (e.g. with keyword `reduce`)
  - vertical aggregation (e.g. with keyword `count`, `max`)
- `CREATE`/`DELETE`/`SET`: update the property graph

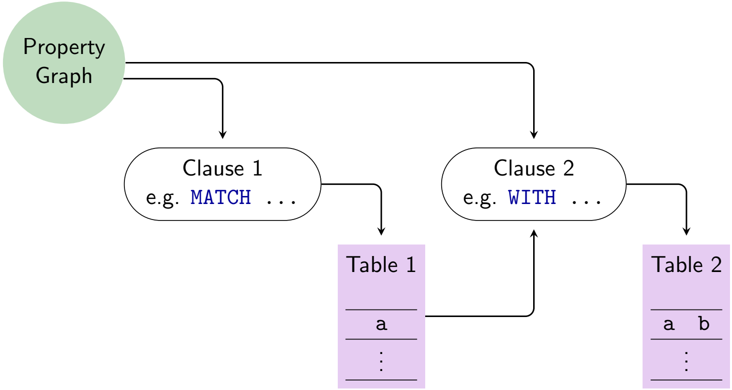
A Cypher query actually chain clauses



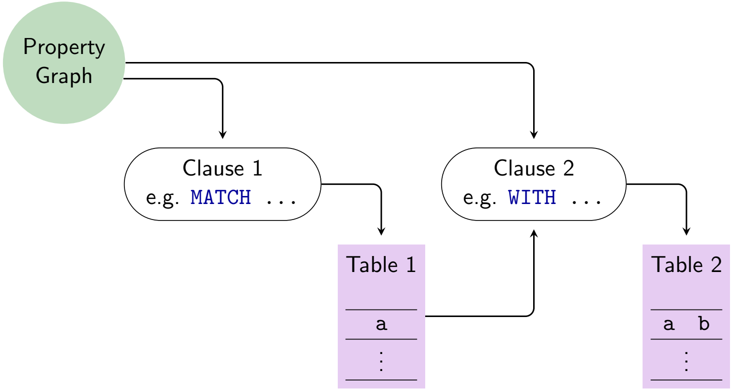
A Cypher query actually chain clauses



A Cypher query actually chain clauses

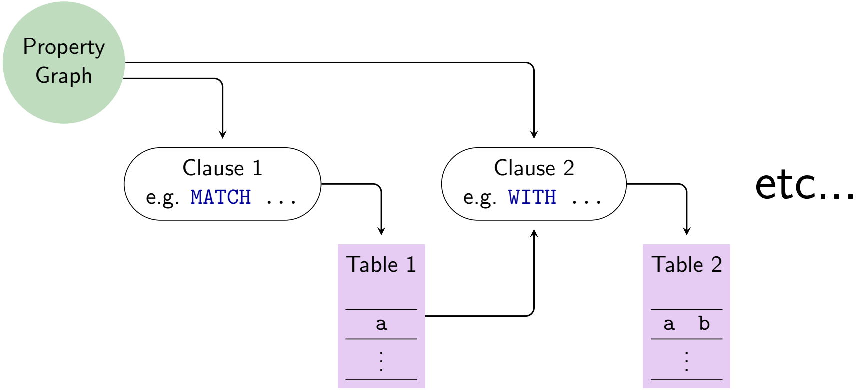


A Cypher query actually chain clauses



etc...

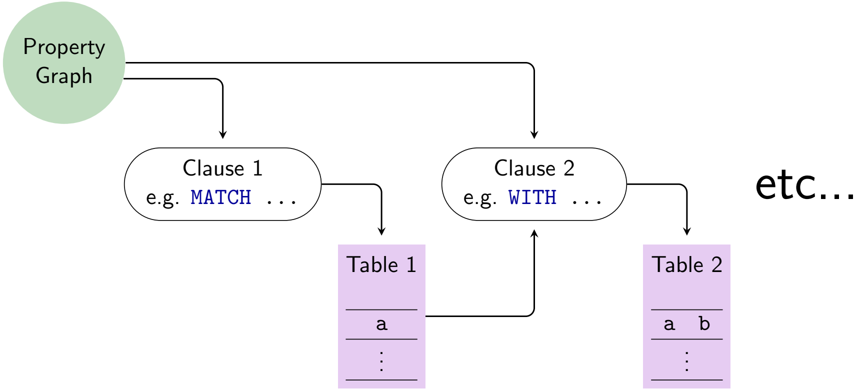
A Cypher query actually chain clauses



Example

- Clause 1 makes some pattern matching
- Clause 2 aggregates over the result of Clause 1

A Cypher query actually chain clauses



Example

- Clause 1 makes some pattern matching
- Clause 2 aggregates over the result of Clause 1

$\Rightarrow$ Trail semantics (rich post-processing at the cost of efficiency)

GQL, standard query language for property graphs

[Deutsch et al. 2022][Francis et al. 2023]

GQL, standard query language for property graphs

[Deutsch et al. 2022][Francis et al. 2023]

Features inherited from Cypher

- ASCII-art syntax
- Graph-to-tables
- Chaining of clauses
- Compute walks

GQL and the fundamental challenge with RPQs

An RPQ may have infinitely many matches

- GQL has to ensure finiteness of answer
- No solution is clearly superior

GQL and the fundamental challenge with RPQs

An RPQ may have infinitely many matches

- GQL has to ensure finiteness of answer
- No solution is clearly superior

GQL does not choose

- Trail semantics $\rightarrow$ keyword `TRAIL`
- Shortest-walk semantics $\rightarrow$ keyword `SHORTEST`
- Syntax restriction $\rightarrow$ keyword `WALK`
- Mixing semantics

Going forward

Study computational problems

- Distinct enumeration under run-based semantics
- Fine-grain complexity of problems is mostly open
- Extend the model to add data
- Usage of multiple semantics

Going forward

Study computational problems

- Distinct enumeration under run-based semantics
- Fine-grain complexity of problems is mostly open
- Extend the model to add data
- Usage of multiple semantics

Thinking outside the box

- Coverage
- Theoretical framework to compare RPQ semantics
- Semantics that output something other than walks

Going forward

Study computational problems

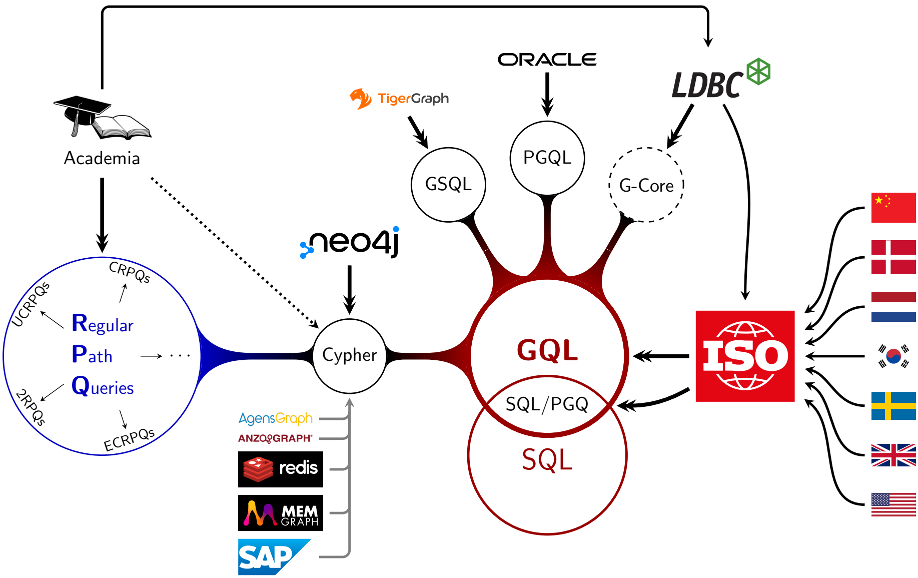
- Distinct enumeration under run-based semantics
- Fine-grain complexity of problems is mostly open
- Extend the model to add data
- Usage of multiple semantics

Thinking outside the box

- Coverage
- Theoretical framework to compare RPQ semantics
- Semantics that output something other than walks

The dream

- Add run-based semantics to GQL 2.0



Thank you for your attention!

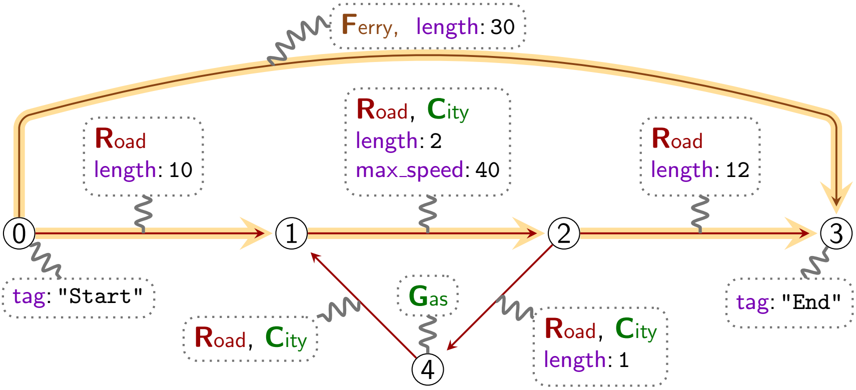
Bibliography I

- Angles, Arenas, Barceló, Boncz, Fletcher, Gutierrez, Lindaaker, Paradies, Plantikow, Sequeda, Rest, and Voigt (2018). "G-CORE: A Core for Future Graph Query Languages". In: *SIGMOD'18*. ACM, pp. 1421–1432.
- Angles, Arenas, Barceló, Hogan, Reutter, and Vrgoč (2017). "Foundations of Modern Query Languages for Graph Databases". In: *ACM Comput. Surv.* 50.5.
- Cruz, Mendelzon, and Wood (1987). "A Graphical Query Language Supporting Recursion". In: *SIGMOD'87*. ACM, pp. 323–330.
- David, Francis, and Marsault (2023). "Run-Based Semantics for RPQs". In: *KR'23*.
- Deutsch, Francis, et al. (2022). "Graph Pattern Matching in GQL and SQL/PGQ". In: *SIGMOD'22*.
- Francis, Gheerbrant, Guagliardo, Libkin, Marsault, Martens, Murlak, Peterfreund, Rogova, and Vrgoč (2023). "GPC: A Pattern Calculus for Property Graphs". In: *PODS'23*.

Bibliography II

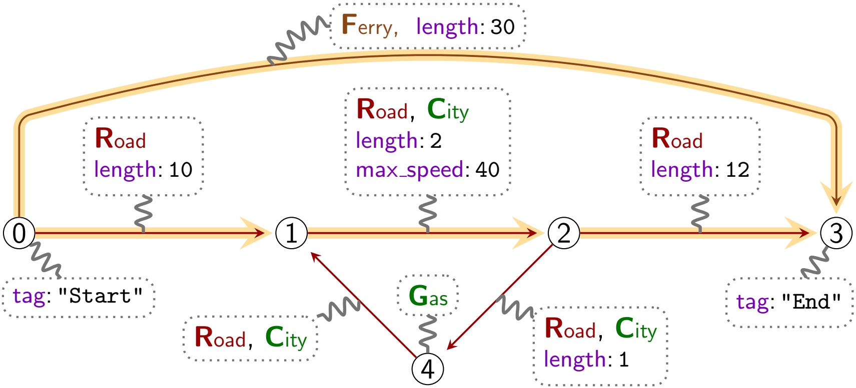
- Francis, Green, Guagliardo, Libkin, Lindaaker, Marsault, Plantikow, Rydberg, Selmer, and Taylor (2018). "Cypher: An Evolving Query Language for Property Graphs". In: *SIGMOD'18*. ACM.
- International Organization for Standardization (2023). *SQL – Part 16: SQL Property Graph Queries (SQL/PGQ)*. Standard ISO/IEC CD 9075-16.2. URL: `https://www.iso.org/standard/79473.html`.
- ISO (2024). *GQL*. Standard under development ISO/IEC CD 39075. To appear. URL: `https://www.iso.org/standard/76120.html`.
- Oracle (2021). *PGQL 2.0 Specification*. URL: `https://pgql-lang.org/spec/2.0/`.
- TigerGraph (2023). *GSQL Language Reference (version 3.9)*. URL: `https://docs.tigergraph.com/gsql-ref/3.9/intro/`.
- World Wide Web Consortium (2013). *SPARQL 1.1 Query Language, Section 9: Property paths*. `https://www.w3.org/TR/sparql11-query/#propertypaths`.

GQL path-bindings in one slide




```
MATCH TRAIL (a WHERE a.tag="Start")
         [ -[r:Road]-> | -[c:City]-> ]* (b WHERE b.tag="End")
```

GQL path-bindings in one slide




```
MATCH TRAIL (a WHERE a.tag="Start")
         [ -[r:Road]-> | -[c:City]-> ]* (b WHERE b.tag="End")
```

0	→	1	→	2	→	3
a	r		r		r	b

0	→	1	→	2	→	3
a	r		c		r	b