

Introduction to querying with RPQs

semantics in theory and practice

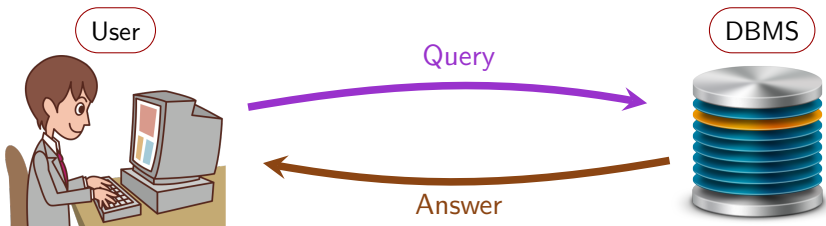
Victor MARSAULT

Université Gustave-Eiffel, CNRS, LIGM

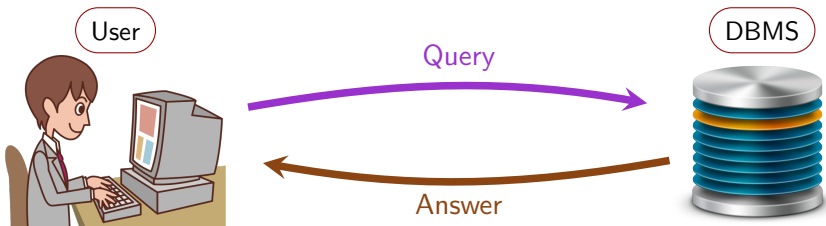
Séminaire Automates, IRIF

2023-09-22

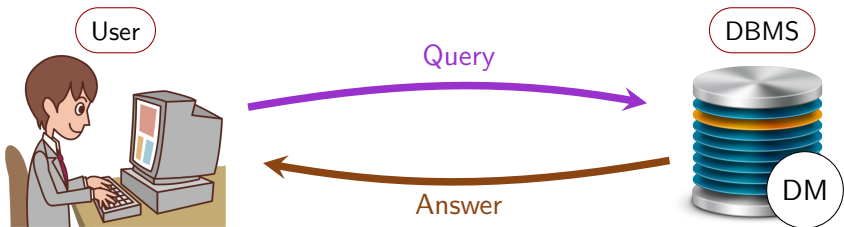
Introduction



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

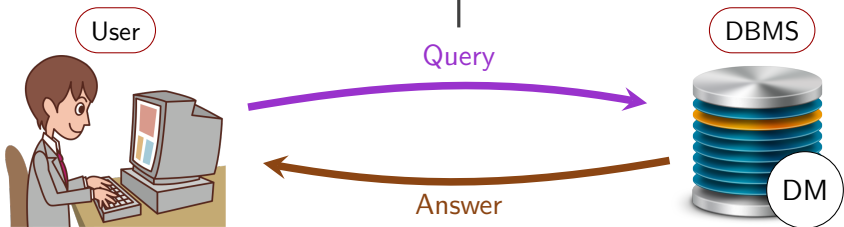


- DBMS = **D**ata**B**ase **M**anagement **S**ystem



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

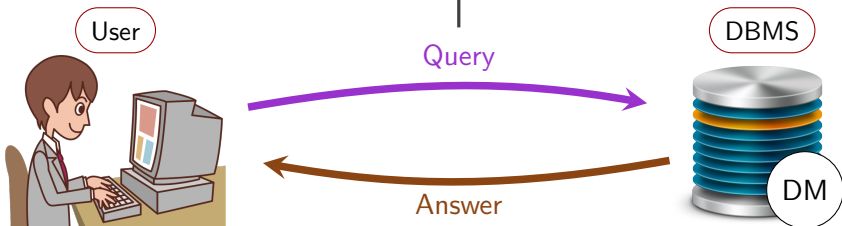
- DBMS = **D**ata**B**ase **M**anagement **S**ystem
- Query language
 - "What can users ask for?"



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

- 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

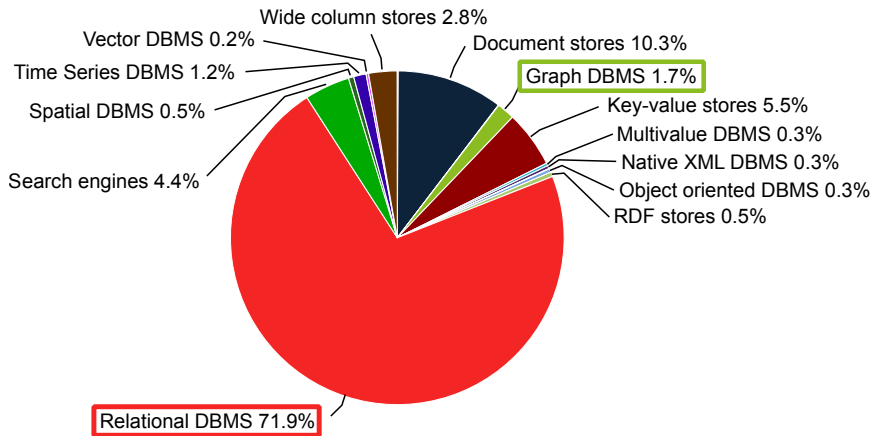


Figure and data from db-engines.com, August 2023

Graph DBMS is growing in popularity

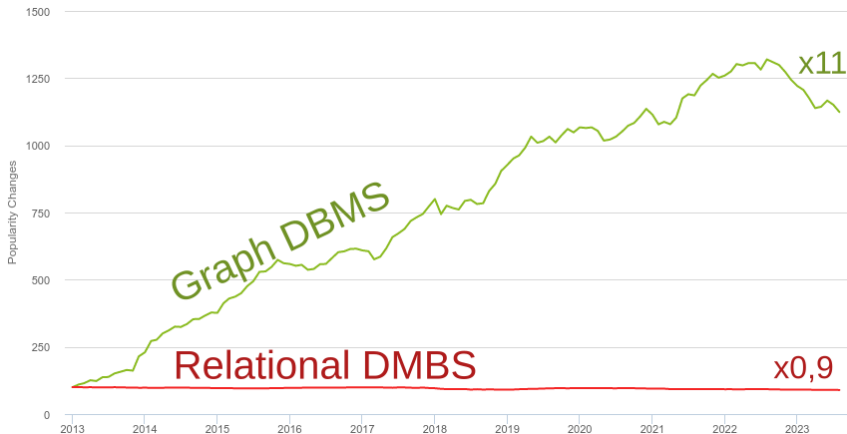


Figure and data from db-engines.com, August 2023

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

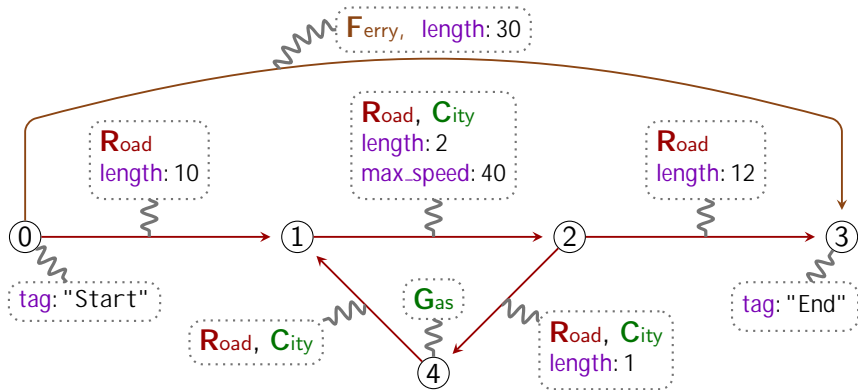
id	buyer	date
0	Alice	01-11-1865
1	Bob	07-07-2022

Order-content table

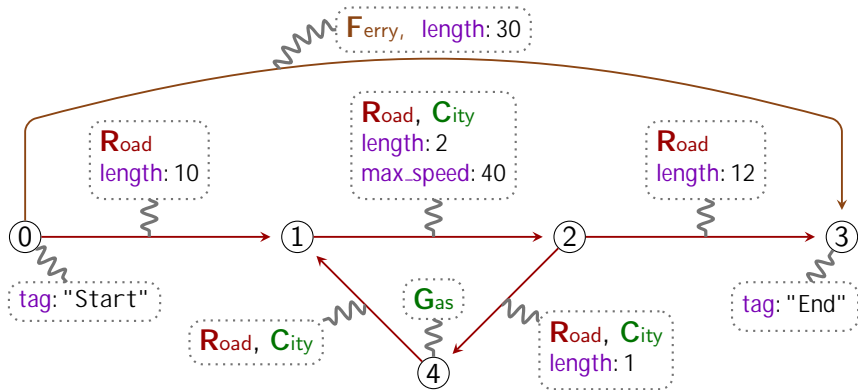
order_id	product
0	Rabbit
0	Pocket Watch
1	Sponge
1	Broom

Why use graphs ?

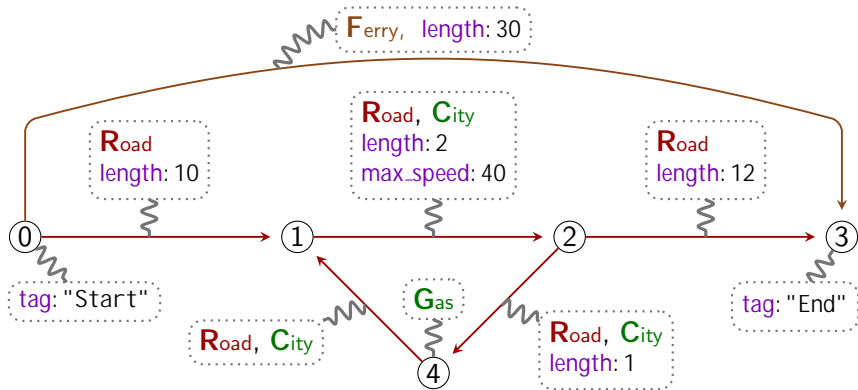
Some data have intrinsically the structure of graphs (e.g. networks)



Why not store graphs in tables?

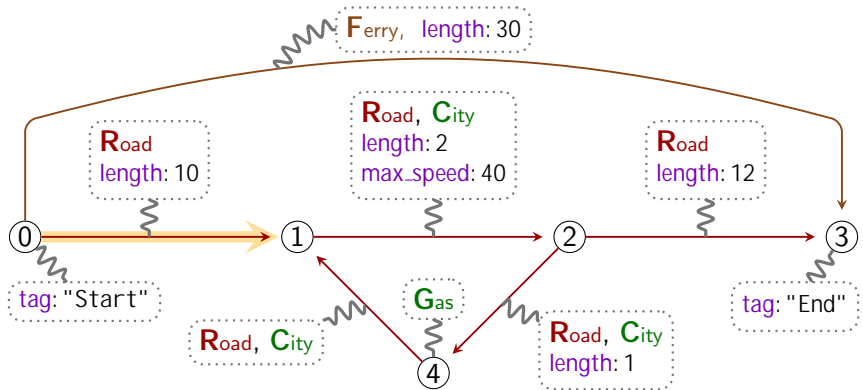


Why not store graphs in tables?



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		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

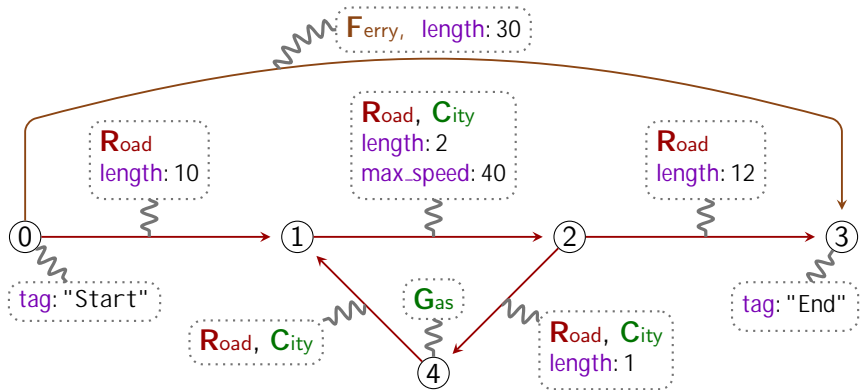
Why not store graphs in tables?



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

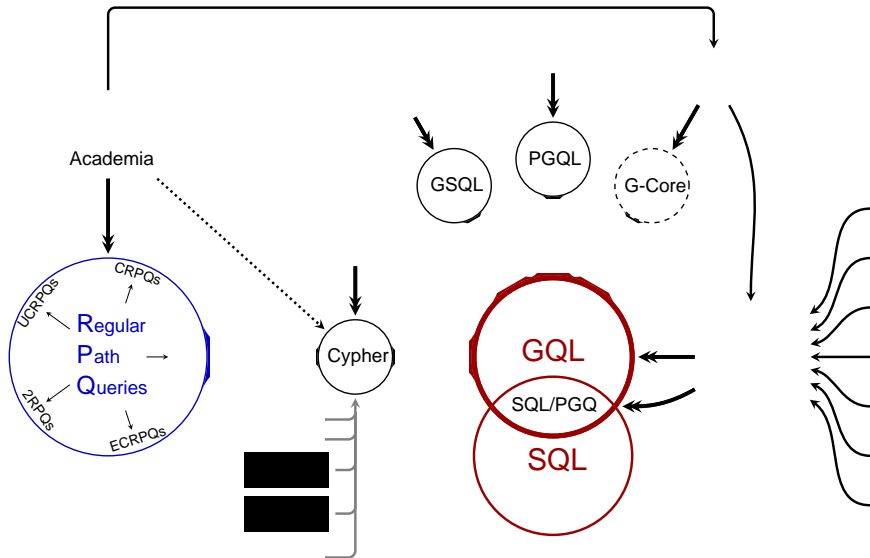


id	source_id	target_id	Road	Ferry	City	length	max_speed
e ₀₁	0	1	true	false	false	10	
e ₁₂	1	2	true	false	true	10	40
e ₂₄	2	4	true	false	true		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

History of query languages for property graphs

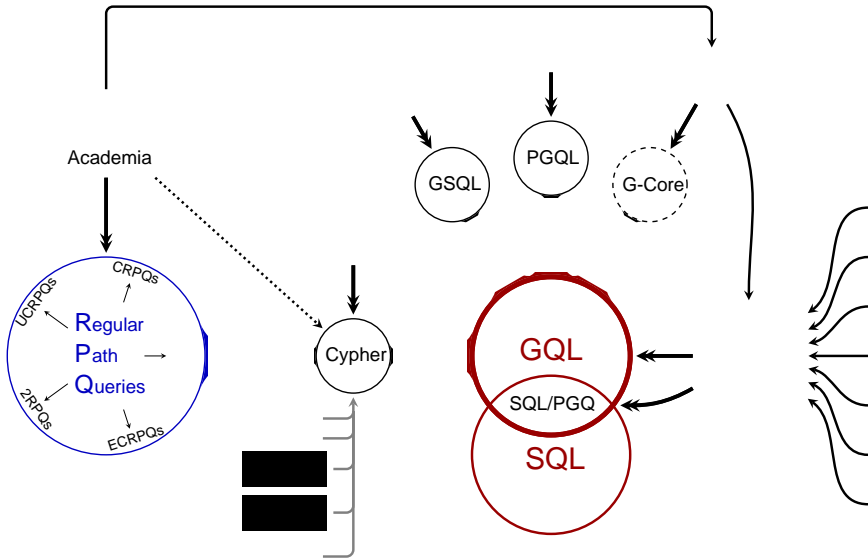
From RPQs to GQL: history and actors

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



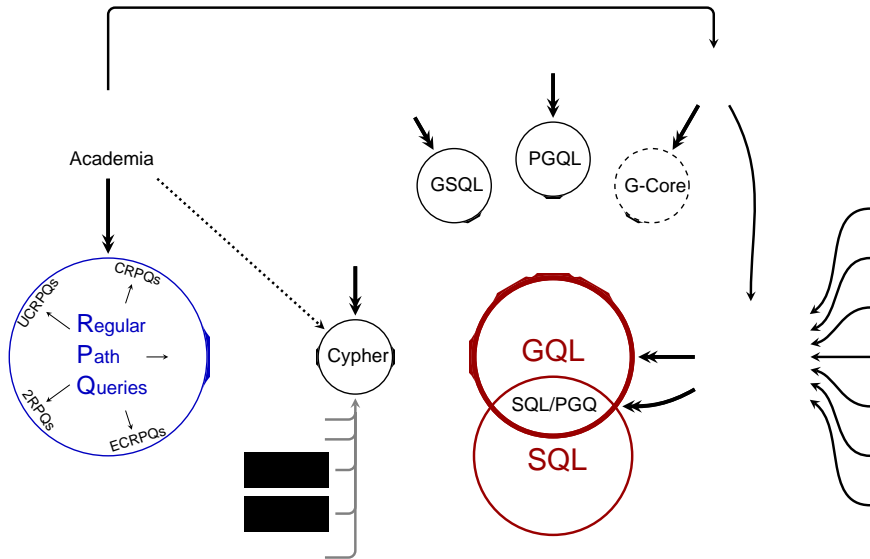
From RPQs to GQL: history and actors

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



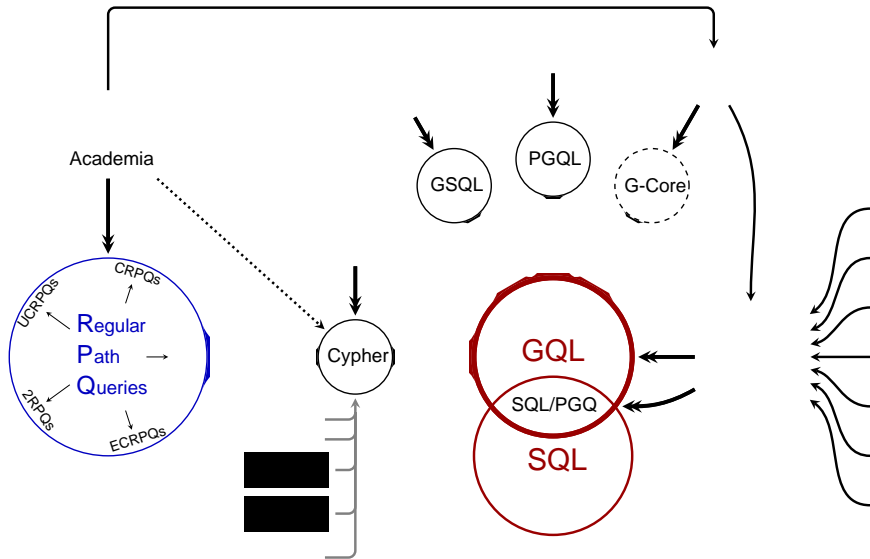
From RPQs to GQL: history and actors

2011 { Cypher is designed by Neo4j



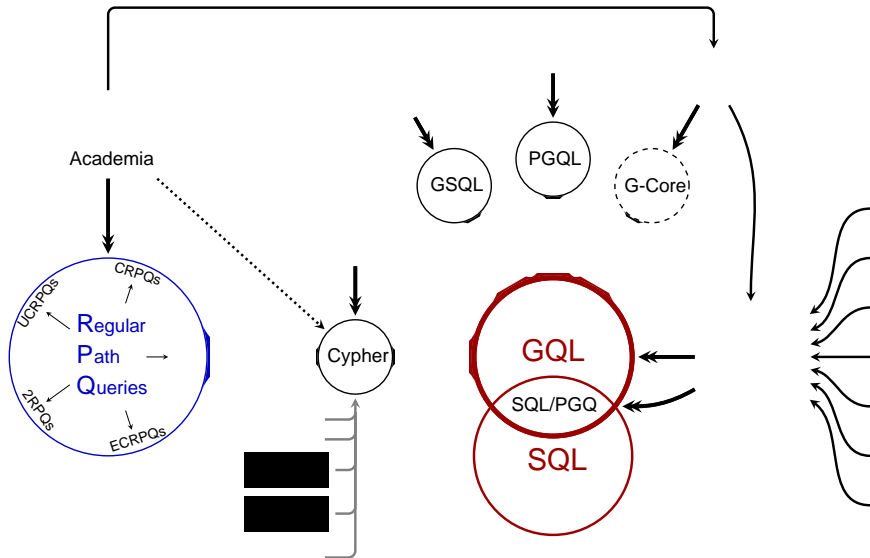
From RPQs to GQL: history and actors

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



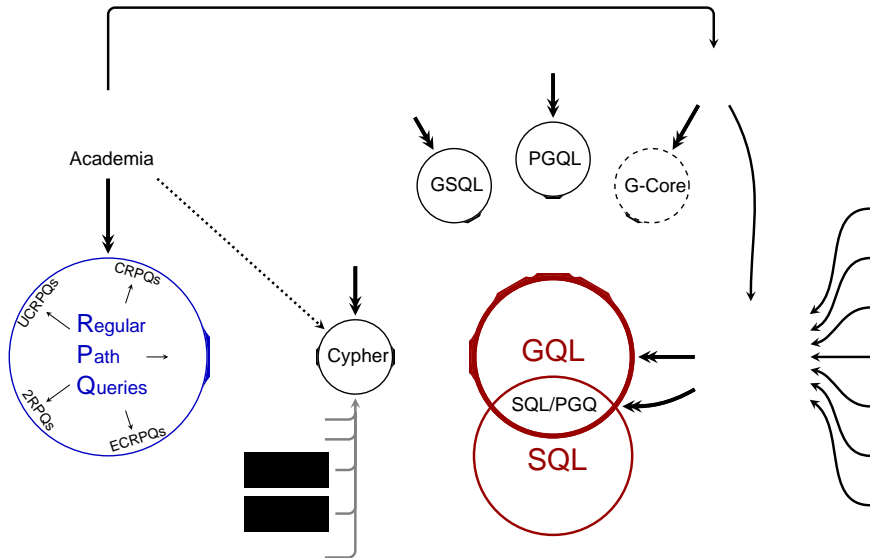
From RPQs to GQL: history and actors

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



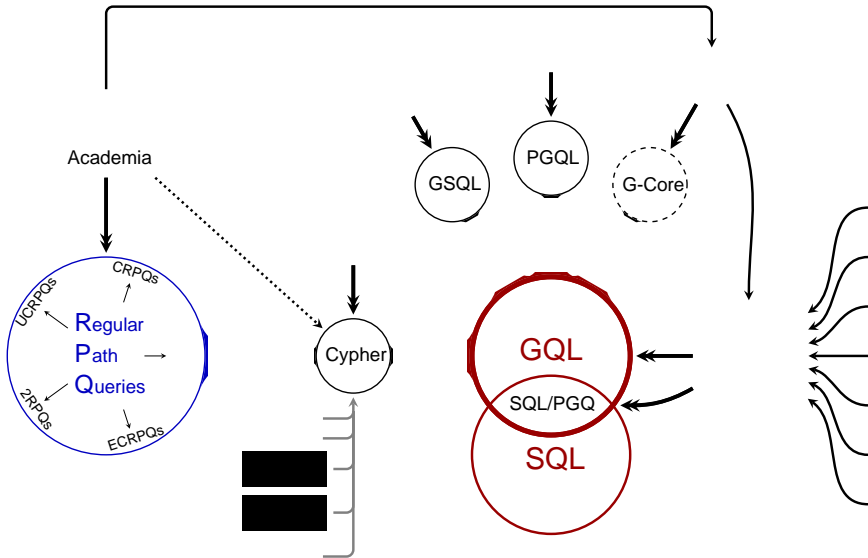
From RPQs to GQL: history and actors

mid 2010's { Other languages/DBMS are released



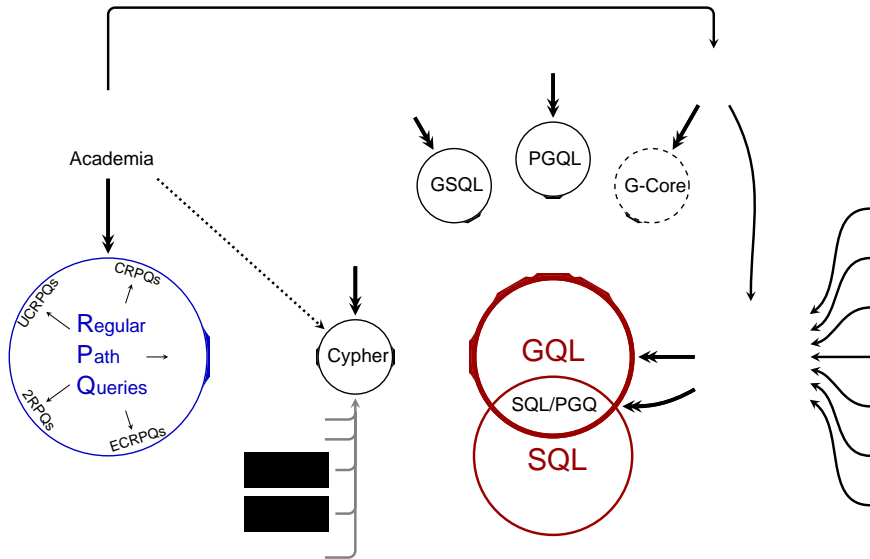
From RPQs to GQL: history and actors

late 2010's { Merge all existing languages ?



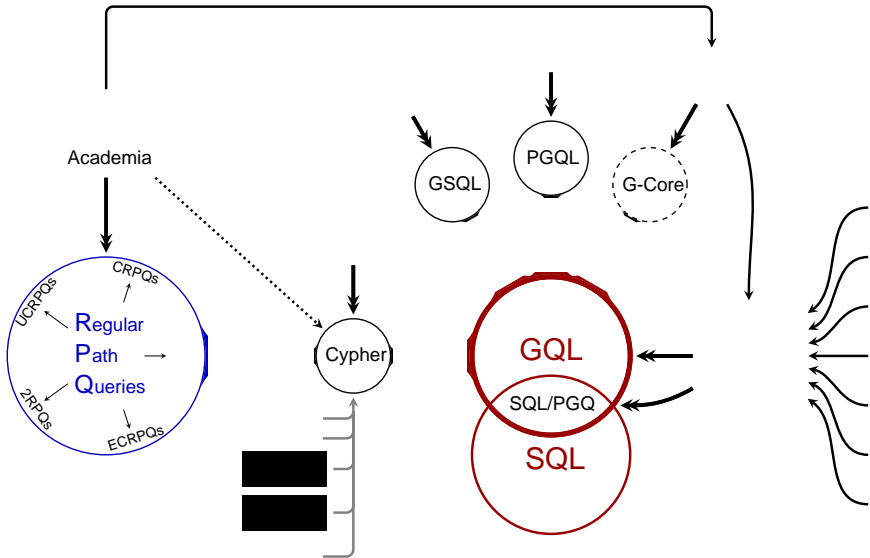
From RPQs to GQL: history and actors

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



From RPQs to GQL: history and actors

2024 (expected) { Publication of version 1 of GQL



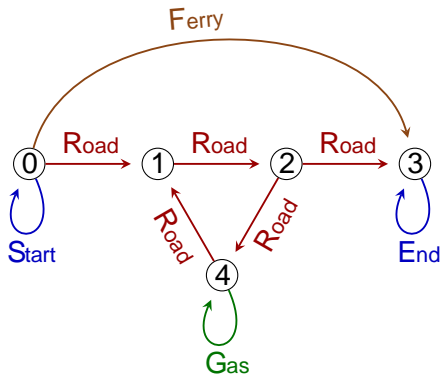
Foundation of querying graph databases: RF

A graph consists of ...

- Vertices (or Nodes)
- Edges (or Relationships)
- Edge labels: **R**; **F**; **G**; **S**; **Eg**

Walk

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



A graph consists of ...

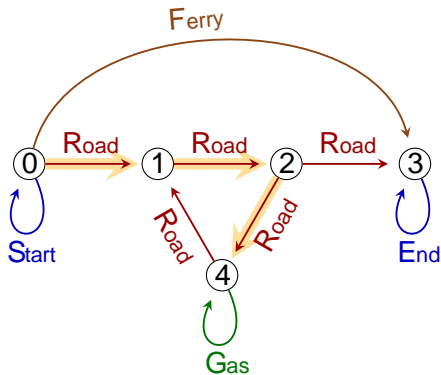
- Vertices (or Nodes)
- Edges (or Relationships)
- Edge labels $R; F; G; S; E$

Walk

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

0! 1! 2! 4

is labeled by RRR



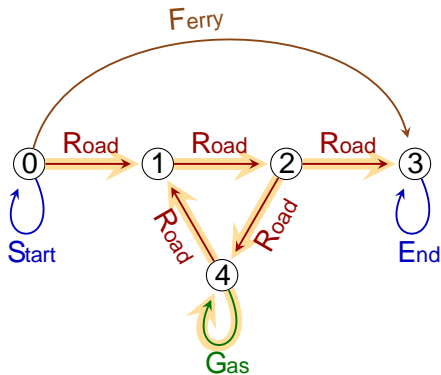
A graph consists of ...

- 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! 1! 2! 4! 4! 1! 2! 3
is labeled by **RRRGRRR**



$Q ::=$ A
 QQ
 Q + Q
 Q

where A is a label in the graph.

$Q ::=$ A
 QQ
 Q + Q
 Q

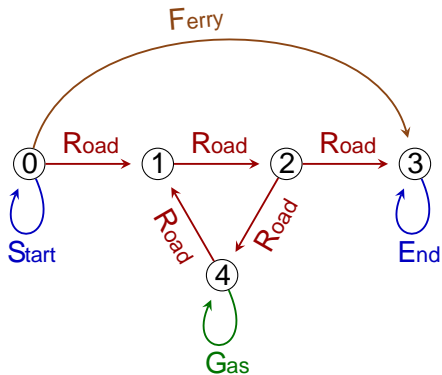
where A is a label in the graph.

Matches

A match for Q is any walk w such that Q denotes the label of w

QueryRR matches...

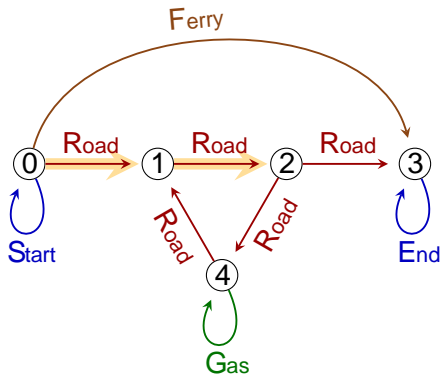
...walks of two Road edges



QueryRR matches...

...walks of two Road edges

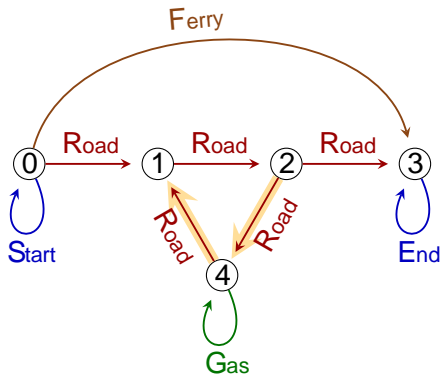
0!	1!	2	2!	4!	1
1!	2!	3	4!	1!	2
1!	2!	3			



QueryRR matches...

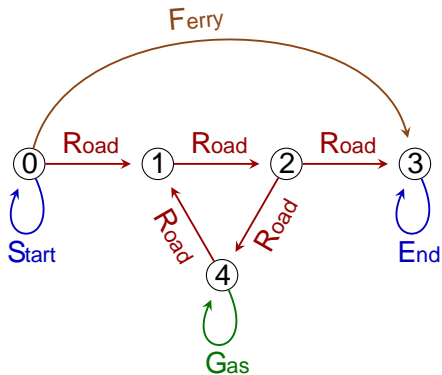
...walks of two Road edges

0!	1!	2	2!	4!	1
1!	2!	3	4!	1!	2
1!	2!	3			



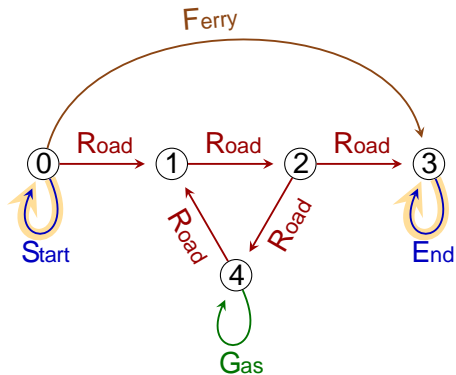
$Q_1 = S(R+F) E$

Q_1 matches...



$Q_1 = S(R+F)E$

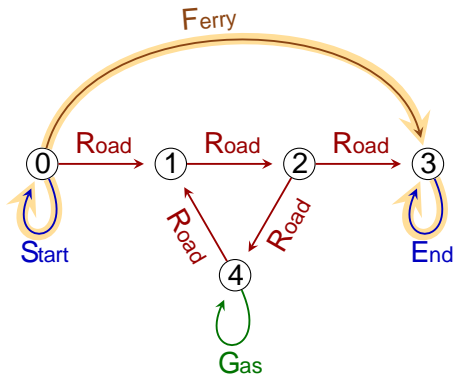
Q_1 matches...



$Q_1 = S(R+F) E$

Q_1 matches...

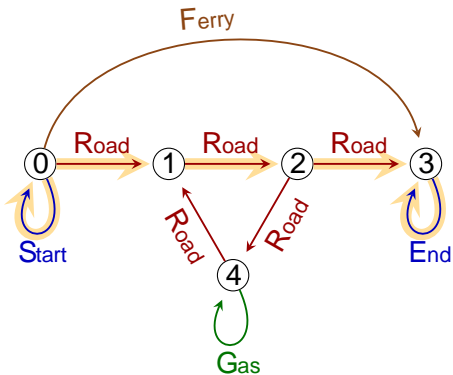
- The ferry



$Q_1 = S(R+F) E$

Q_1 matches...

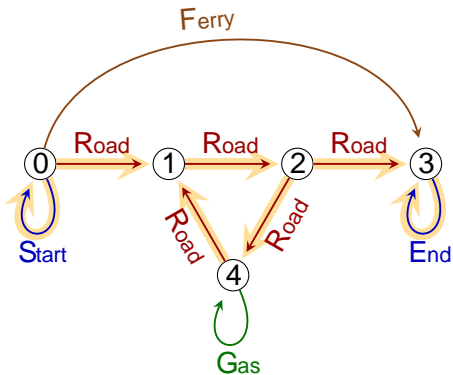
- The ferry
- The direct road



$Q_1 = S(R+F) E$

Q_1 matches...

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



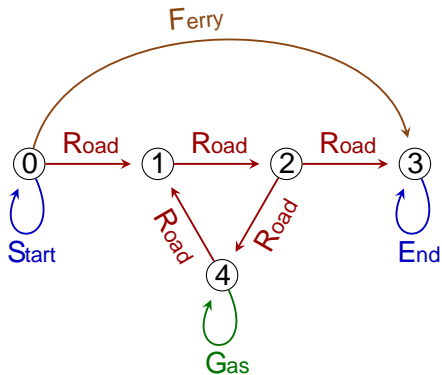
$Q_1 = S(R+F) E$

Q_1 matches...

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

$Q_2 = S(R+F) G(R+F) E$

Q_2 matches...



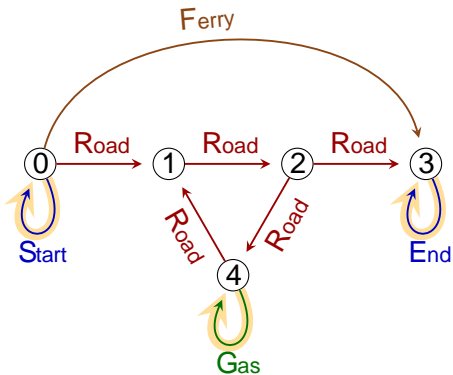
$Q_1 = S(R+F) E$

Q_1 matches...

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

$Q_2 = S(R+F) G(R+F) E$

Q_2 matches...



$Q_1 = S(R+F) E$

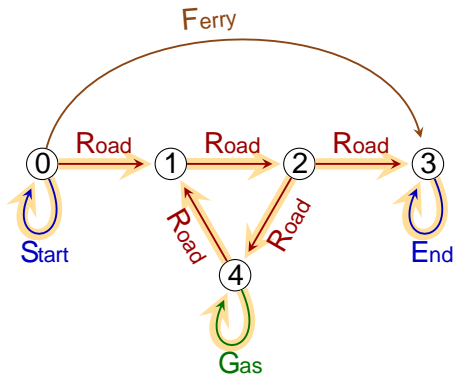
Q_1 matches...

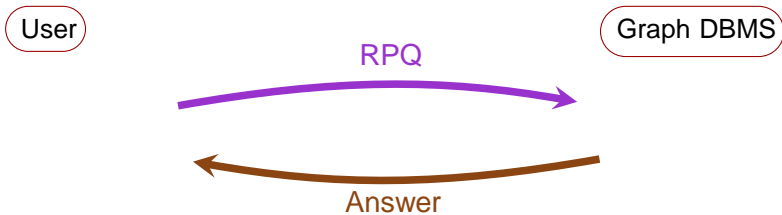
- The ferry
- The direct road
- Roads with laps in the circuit

$Q_2 = S(R+F) G(R+F) E$

Q_2 matches...

- Roads with laps in the circuit





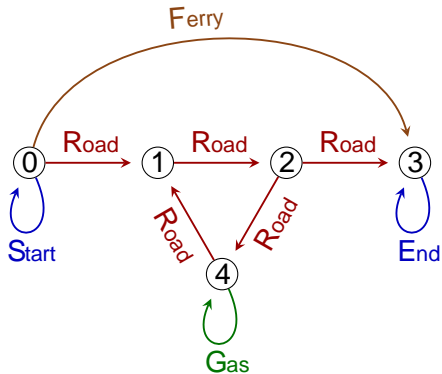
⚠ In nitely many matches but nite answer! ⚠

Semantics of RPQs

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

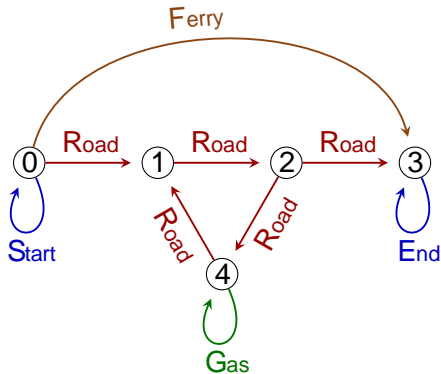
$Q_1 = S (R+F) E$

$Q_2 = S (R+F) G (R+F) E$

- All matches are of the form:

$0! \quad ! \quad 3$

) Q_1 and Q_2 return $f(0; 3)g$



Pros and cons

Pros

- Efficient algorithms
- Well grounded theory

Pros and cons

Pros

- Efficient algorithms
- Well grounded theory

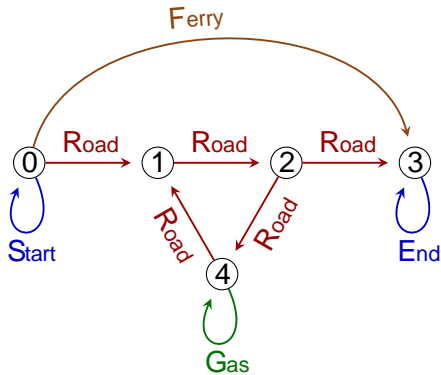
Cons

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

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

Definition

- Return the walk with the least number of edges



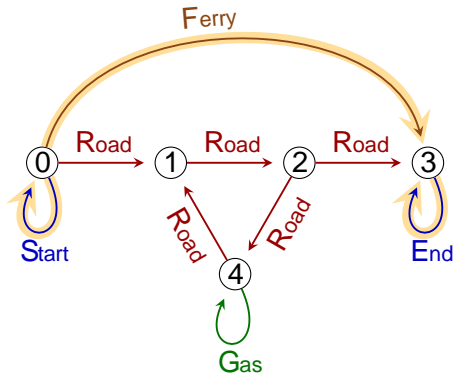
Used in PGQL (Oracle), GSQL (TigerGraph) and G-core [Angles et al. 2018]

Definition

- Return the walk with the least number of edges

$Q_1 = S(R+F)E$

- Q_1 returns 1 walk
 - the ferry
- Walks taking the road have more edges



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

Definition

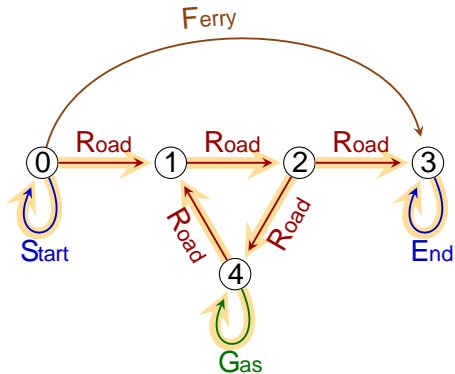
- Return the walk with the least number of edges

$Q_1 = S(R+F)E$

- Q_1 returns 1 walk
 - the ferry
- Walks taking the road have more edges

$Q_2 = S(R+F)G(R+F)E$

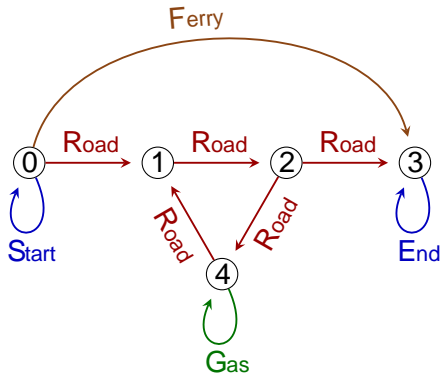
- Q_1 returns 1 walk
 - the one-lap road



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

Definition

- Return walks
- Forbid to repeat edges



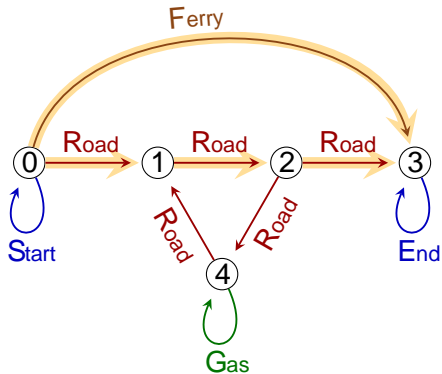
Used in Cypher (Neo4j) [Francis et al. 2018]

Definition

- Return walks
- Forbid to repeat edges

$Q_1 = S (R + F) E$

- Q_1 returns 2 walks
 - the ferry
 - the straight road



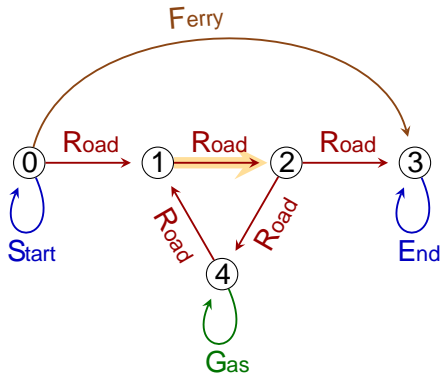
Used in Cypher (Neo4j) [Francis et al. 2018]

Definition

- Return walks
- Forbid to repeat edges

$Q_1 = S(R+F)E$

- 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

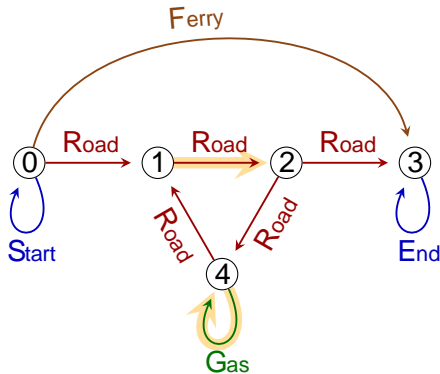
- Return walks
- Forbid to repeat edges

$$Q_1 = S (R + F) E$$

- Q_1 returns 2 walks
 - the ferry
 - the straight road
- Walks with circuit laps repeat the middle edge

$$Q_2 = S (R + F) G (R + F) E$$

- Q_2 returns nothing



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

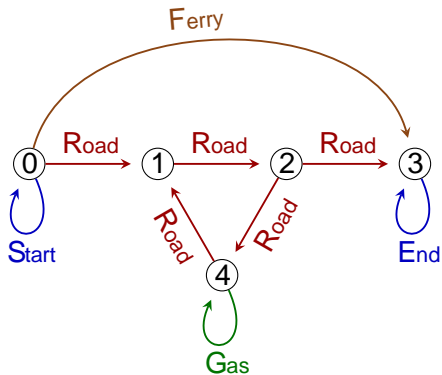
$Q_1 = S(R+F)E$

Shortest-walk semantics

- outputs the **Ferry**-walk only
- computes the "best" answer

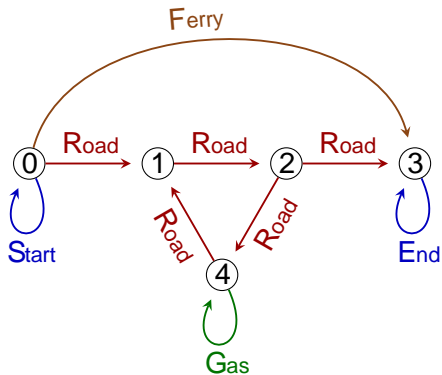
Trail semantics:

- outputs the **Ferry**-walk and the direct **Road**
- 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

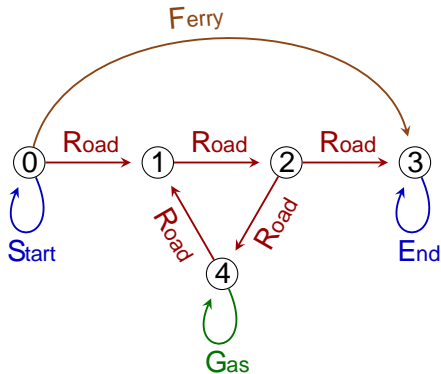
$\begin{matrix} W \\ W \\ \bullet \end{matrix}$ Enables $\begin{matrix} W \\ W \\ \bullet \end{matrix}$

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

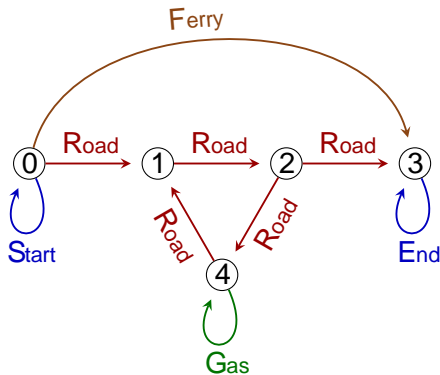
$\begin{matrix} W \\ W \\ \bullet \end{matrix}$ Enables $\begin{matrix} W \\ W \\ \bullet \end{matrix}$

Vertical postprocessing

Compute something across returned walks.

Example:

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



One little problem

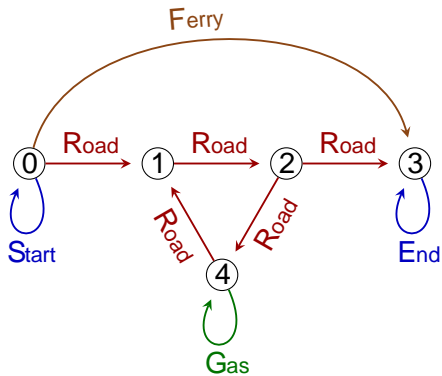
- De ne "coverage"

Fact

Trail sometimes discard walks that seem \nonstupid"

$Q_2 = S (R+F) G (R+F) E$

- Trail outputs nothing

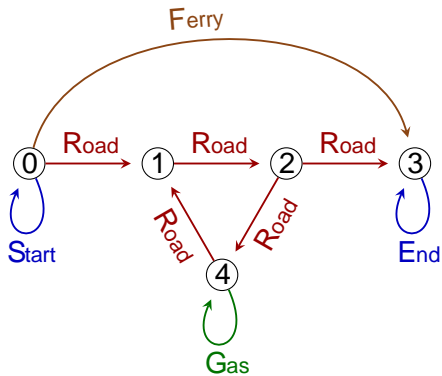


Fact

Trail sometimes discard walks that seem \nonstupid"

$Q_2 = S (R+F) G (R+F) E$

- Trail outputs nothing



Two little problems

- De ne \coverage"
- De ne \good" coverage

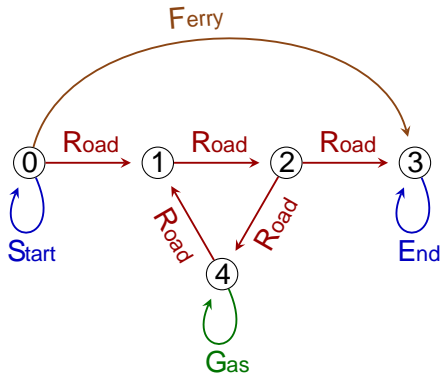
	Shortest-walk	Trail
Existence	■ Tract.	■ Untract.
Enumeration	■ Tract.	■ Untract.
Distinct Enum	■ Tract.	■ Untract.
Membership	■ Tract.	■ Tract.
Counting	■ Meaningless	■ Untract.
Coverage	■ None	■ Some, with no guarantee

Run-based semantics

[David-Francis-Marsault 2023]

Definition

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



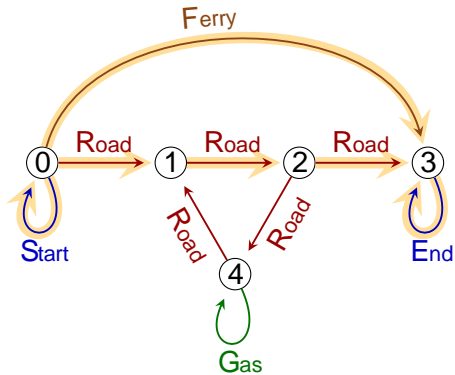
[David-Francis-Marsault 2023]

Definition

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

$$Q_1 = S(R+F)E$$

- Returns
 - the ferry
 - the straight road



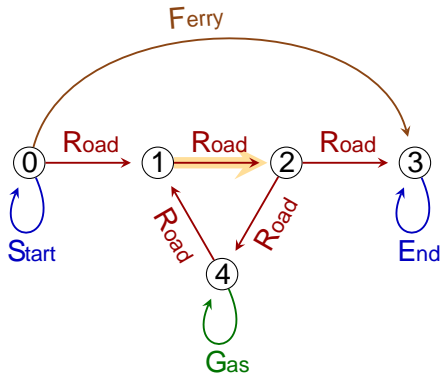
[David-Francis-Marsault 2023]

Definition

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

$$Q_1 = S(R+F)E$$

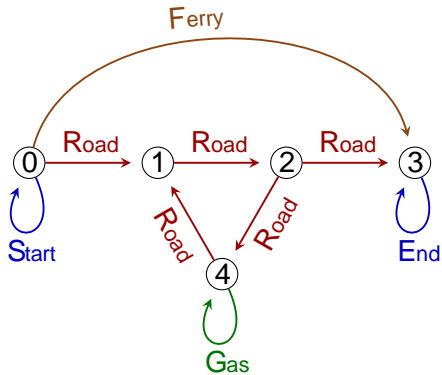
- Returns
 - the ferry
 - the straight road
- In walks with circuit laps
 - ⇒ the middle edge reuses ~~R~~



[David-Francis-Marsault 2023]

Definition

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



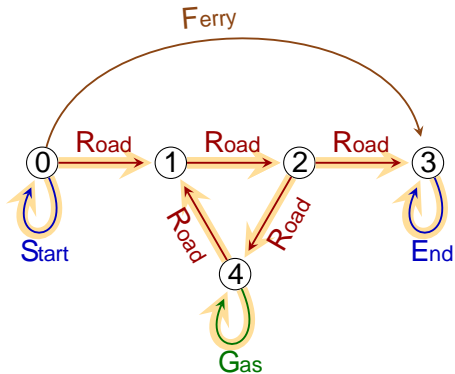
[David-Francis-Marsault 2023]

Definition

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

$Q_2 = S(R+F)G(R+F)E$

- Returns the walk with one circuit lap



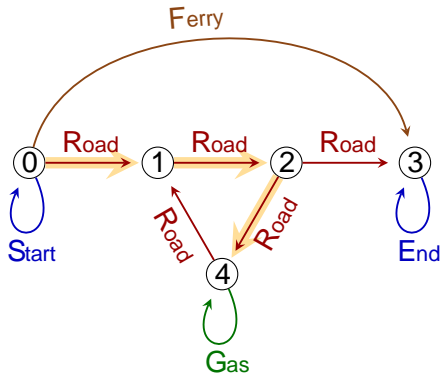
[David-Francis-Marsault 2023]

Definition

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

$Q_2 = S (R+F) G (R+F) E$

- Returns the walk with one circuit lap
 - Before G ! use the left R



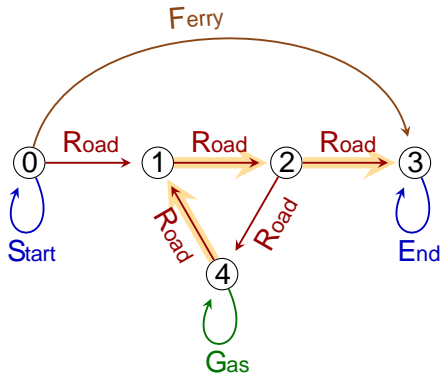
[David-Francis-Marsault 2023]

Definition

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

$Q_2 = S (R + F) G (R + F) E$

- Returns the walk with one circuit lap
 - Before G ! use the left R
 - After G ! use the right R



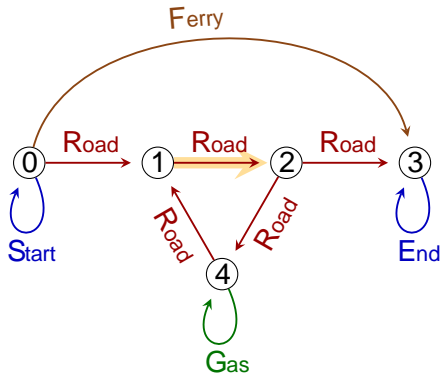
[David-Francis-Marsault 2023]

Definition

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

$Q_2 = S(R+F)G(R+F)E$

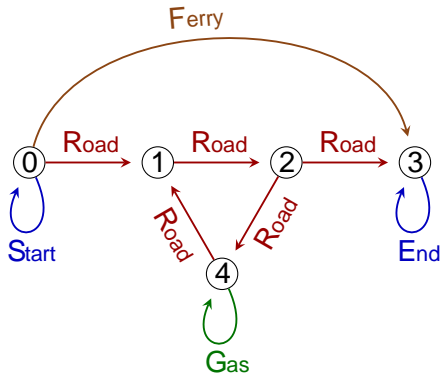
- Returns the walk with one circuit lap
 - Before $G!$ use the left R
 - After $G!$ use the right R
- In walks with 2+ circuit laps
 - =) the middle edge reuses the left R or the right R



Lemma

8 match w of Q

=) some subwalk sw returned



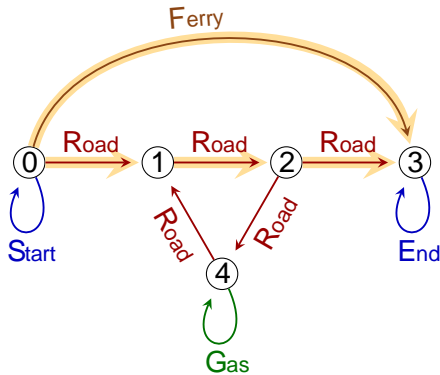
Lemma

8 match w of Q

=) some subwalk sw returned

$Q_1 = S (R + F) E$

- The **Ferry**-walk is returned
- The straight road is returned

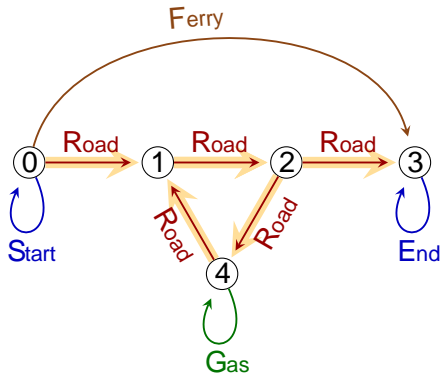


Lemma

8 match w of Q
=> some subwalk sw returned

$$Q_1 = S (R + F) E$$

- The **Ferry**-walk is returned
- The straight road is returned
- Walks with 1 laps are "covered" by the later

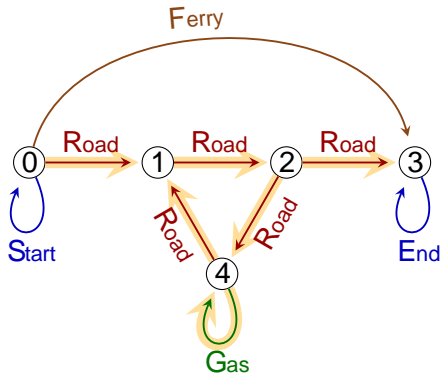


Lemma

8 match w of Q
 => some subwalk sw returned

$$Q_1 = S (R+F) E$$

- The **Ferry**-walk is returned
- The straight road is returned
- Walks with 1 laps are "covered" by the later



$$Q_2 = S (R+F) G (R+F) E$$

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

Binding-trail is syntax-dependent

The output depends on the syntax of the query

R

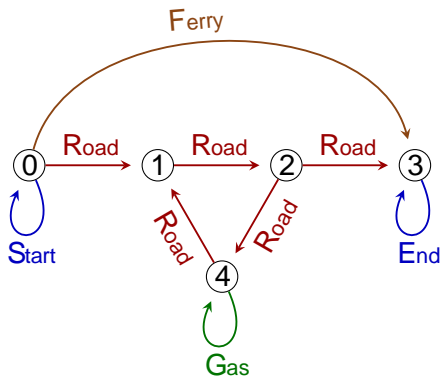
- allows no lap in the circuit

R R

- allows 1 lap in the circuit

(R + R)

- allows 1 lap in the circuit
- In general, \mathbb{R}^n R R



Binding-trail is syntax-dependent

The output depends on the syntax of the query

R

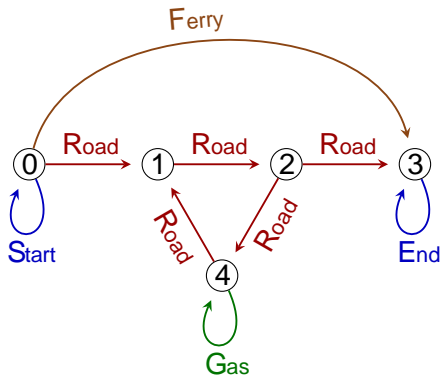
- allows no lap in the circuit

R R

- allows 1 lap in the circuit

(R + R)

- allows 1 lap in the circuit
- In general, \mathbb{R}^n R R

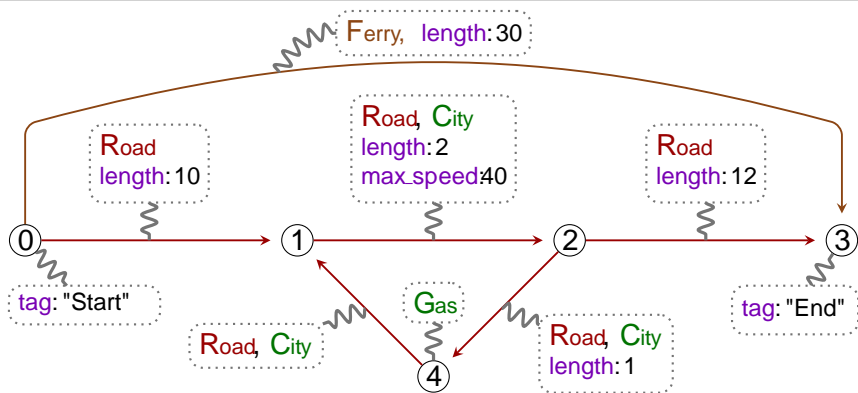


- Unusual from theoretical point of view
- The user has finer control on the output
- This kind of syntax quirks exists in practice

	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 , ...

- 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

- Vertices: MATCH (:Gas)

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

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

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

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

- Vertices: `MATCH (:Gas) MATCH ({tag:"Start"})`
- Edges: `MATCH -[:Road]->`
- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`
- Disjunction: `MATCH ()-[:Road|Ferry]->()`
- Kleene star: `MATCH ()-[:Road*]->()`

- 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]->()`

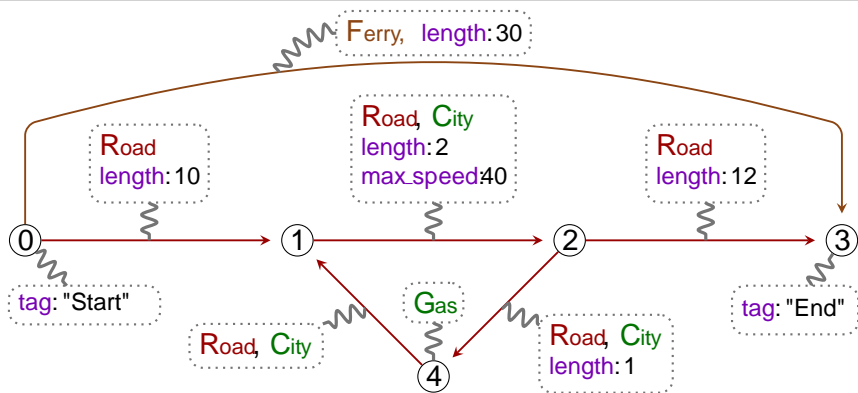
- Vertices: `MATCH (:Gas) MATCH ({tag:"Start"})`
- Edges: `MATCH -[:Road]->`
- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`
- Disjunction: `MATCH ()-[:Road|Ferry]->()`
- Kleene star: `MATCH ()-[:Road*]->()`
- Variables: `MATCH ()-[:Road]->(x)-[:Road]->()`

Cypher queries for Q_1 and Q_2

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

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


Cypher returns a table...



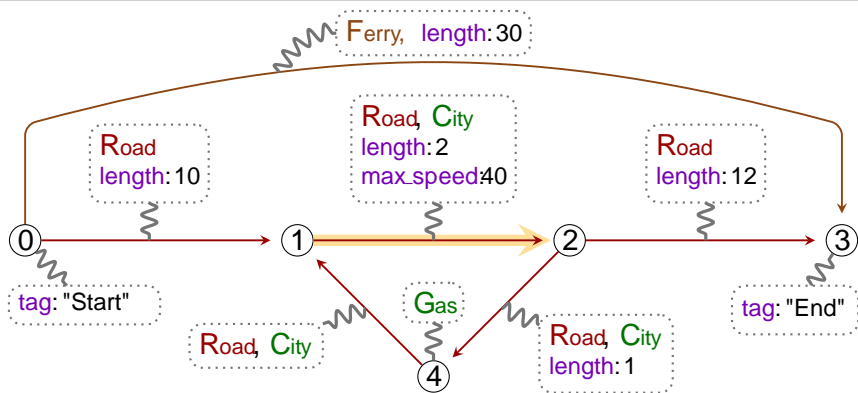
Query

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

Result

s	t
1	2
2	4
4	1

Cypher returns a table...

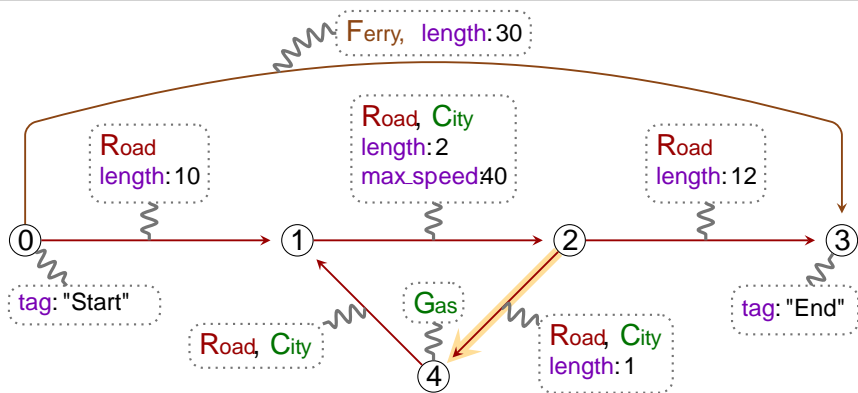


Query

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

Result

s	t
1	2
2	4
4	1



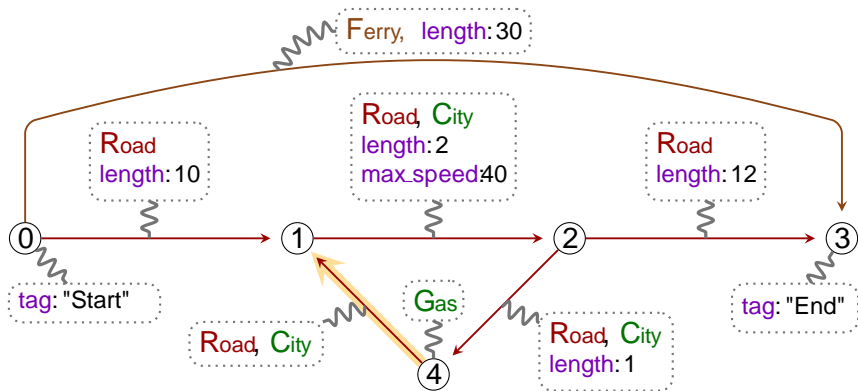
Query

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

Result

s	t
1	2
2	4
4	1

Cypher returns a table...



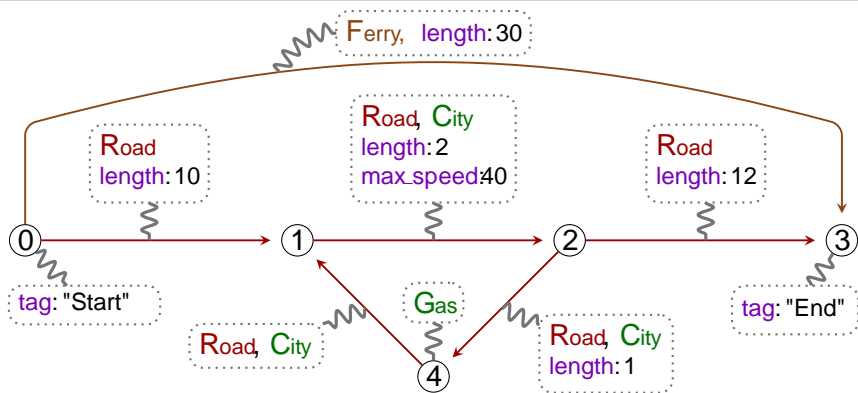
Query

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

Result

s	t
1	2
2	4
4	1

Cypher returns a table... but computes walks



QueryQ₁

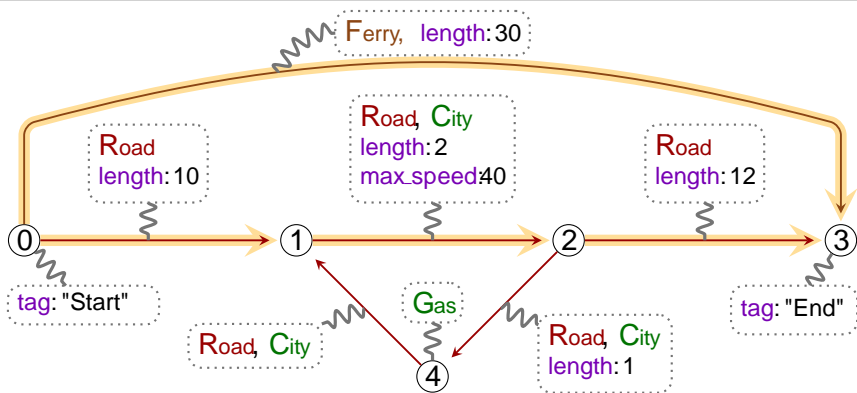
MATCH

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

Result

s	t
0	3
0	3

Cypher returns a table... but computes walks



QueryQ₁

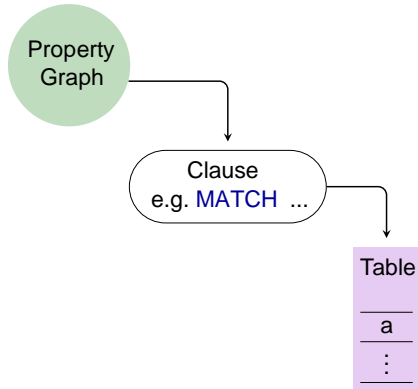
MATCH

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

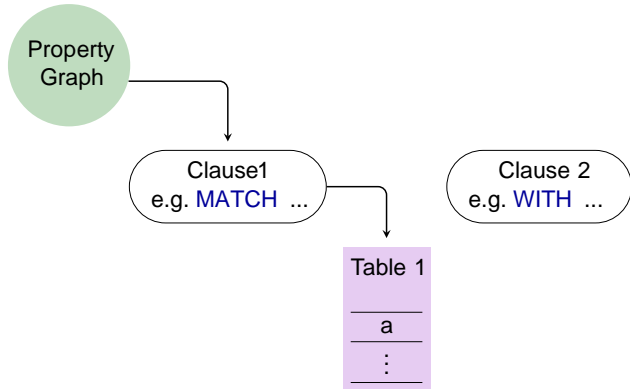
Result

s	t	
0	3	The ferry
0	3	The direct road

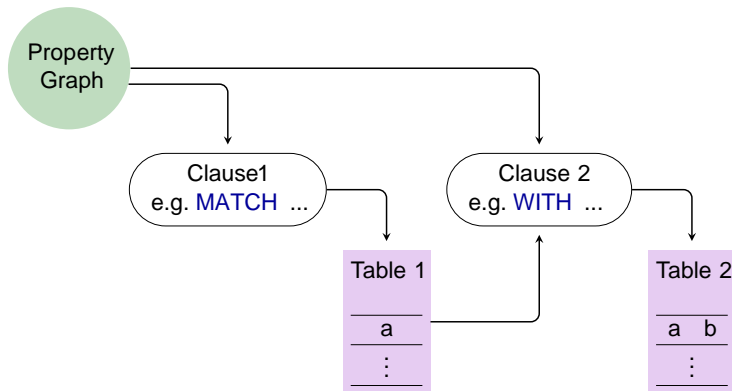
- **WHERE** Iter rows
- **WITH** or **RETURN**
 - add/rename columns
 - horizontal aggregation (e.g. with `keyword_reduce`)
 - vertical aggregation (e.g. with `keyword_count`, `max`)
- **CREATE**/**DELETE**/**SET** update the property graph



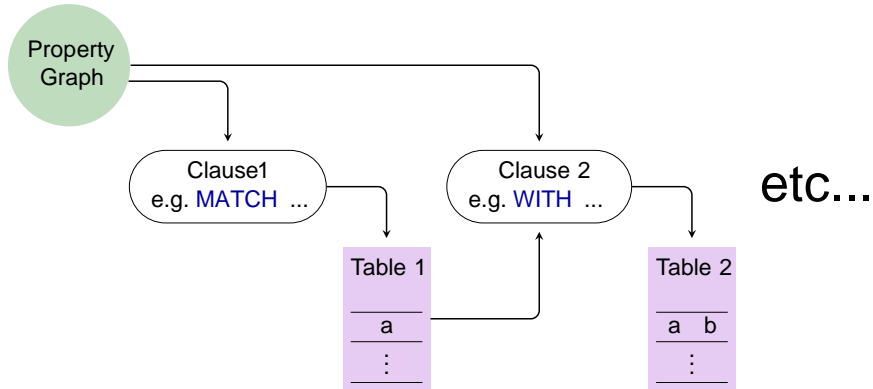
A Cypher query actually chain clauses

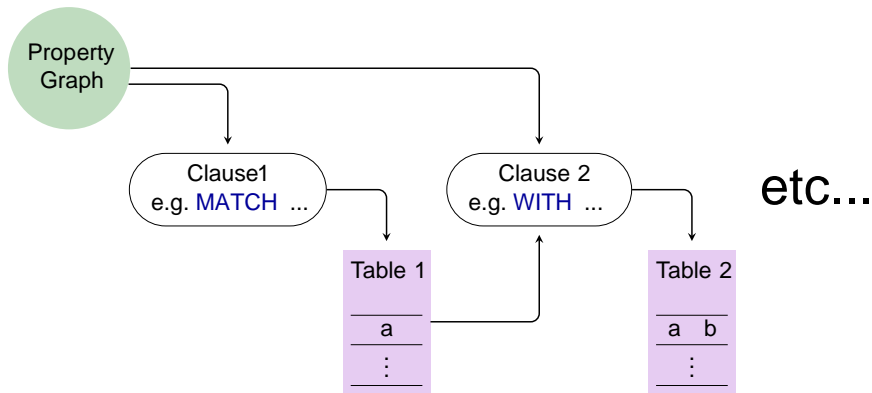


A Cypher query actually chain clauses



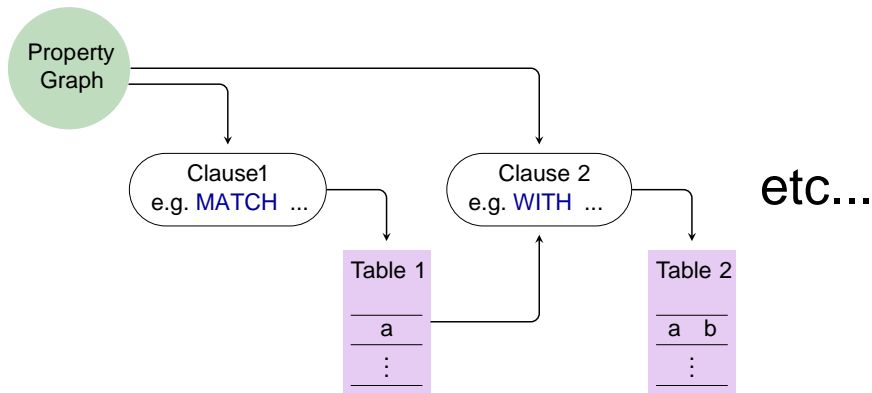
A Cypher query actually chain clauses





Example

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



Example

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

) 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]

Features inherited from Cypher

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

An RPQ may have infinitely many matches

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

An RPQ may have in nitely many matches

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

GQL does not choose

- Trail semantics ! keyword **TRAIL**
- Shortest-walk semantics ! keyword **SHORTEST**
- Syntax restriction ! keyword **WALK**
- Mixing semantics

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

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

Study computational problems

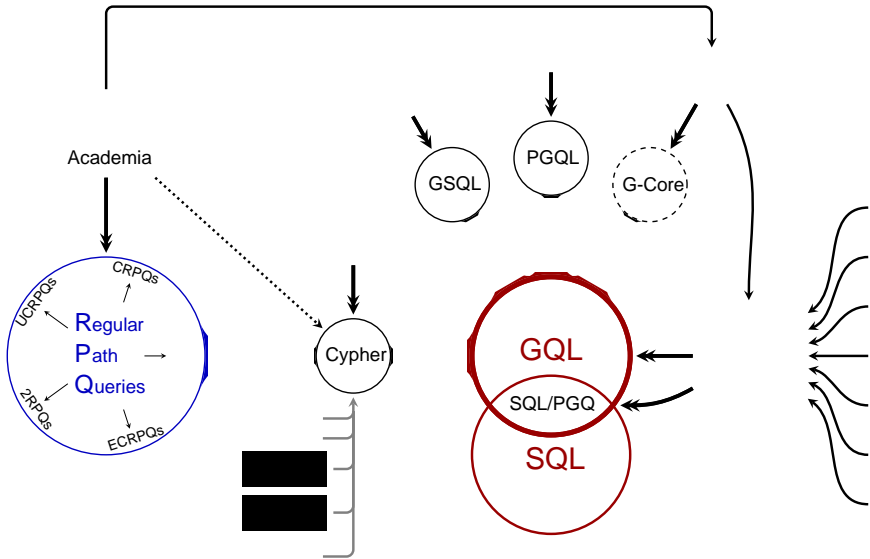
- Distinct enumeration under run-based semantics
- Fine-grain complexity of problems is mostly open
- Extend the model to add data
- Usage of multiple semantics

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!

■ Introduction

- General setting 1
- Relational vs Graph 2
- Relational DBMS 4

■ History of query languages for property graphs

■ Foundation of querying graph databases: RPQs

- Graph as database 8
- RPQ = Regular expression ... 9
- Main queries 11

■ Semantics of RPQs

- Homomorphism semantics .. 13
- Shortest walk 15
- Trail 16
- Coverage 18

■ Run-based semantics







- Definition of binding-trail ... 22
- Coverage 24
- Syntax-dependance 25
- Comparison Tr-SW-BT 26

■ Property graphs and real query languages

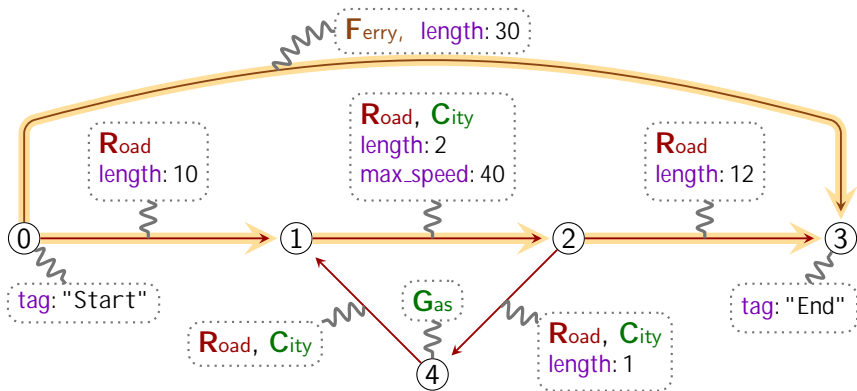
- Example property graph 27
- Cypher 28
- GQL 33

■ Appendix

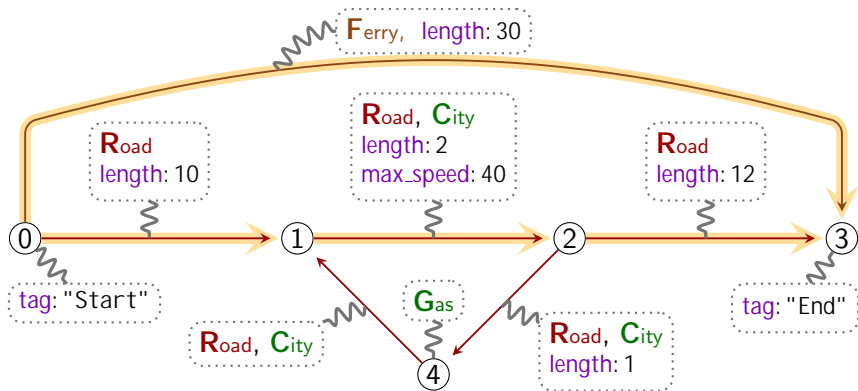
- Bibliography 38
- Path bindings 40

-  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*.

-  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>.



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



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

```
0 / 1 / 2 / 3
a r   r b
```

```
0 / 1 / 2 / 3
a r   c r b
```