Introduction to property graphs:
Data model and query languages

Victor MARSAULT

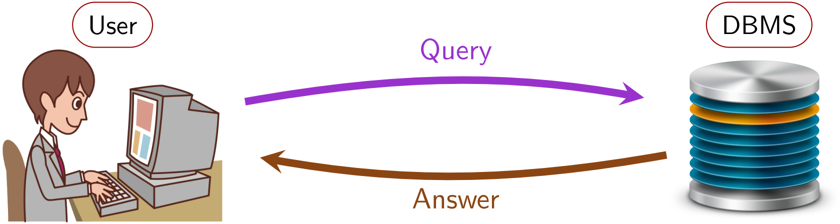
Université Gustave-Eiffel, CNRS, LIGM

MADICS / DOING

2022-07-11

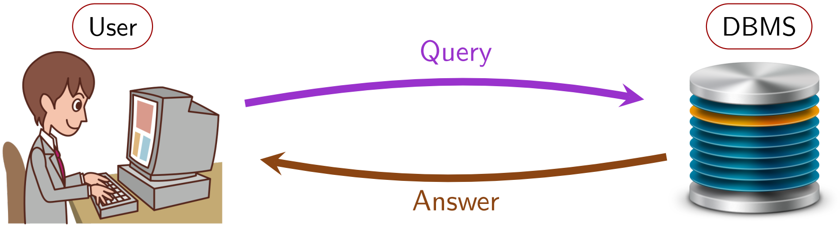
Introduction

General setting



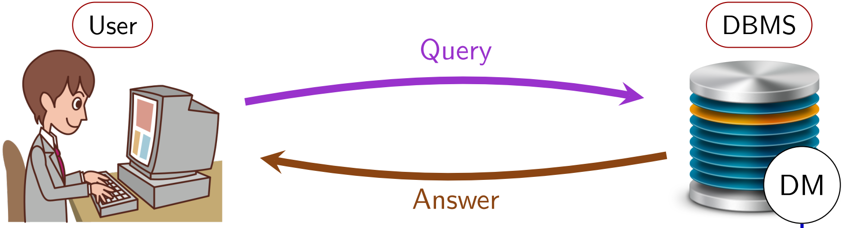
General setting

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



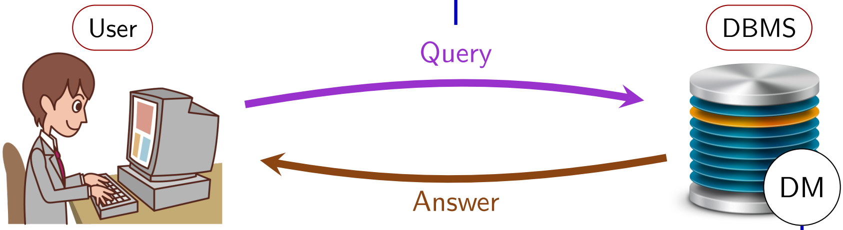
General setting

- 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.

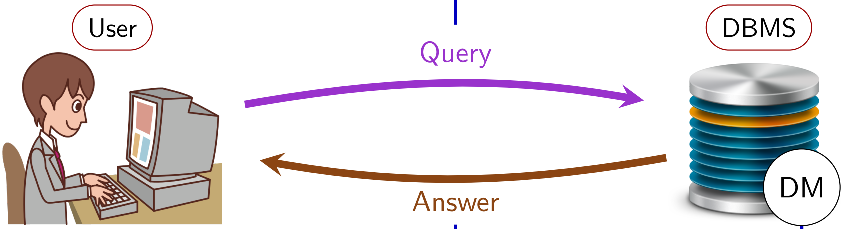
- DBMS = **D**ata**B**ase **M**anagement **S**ystem
- Query language
  - "*What can users ask for?*"



- 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.

Relational DBMS = 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 DBMS = 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 DBMS = 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 DBMS = 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 DBMS = 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 DBMS = 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€

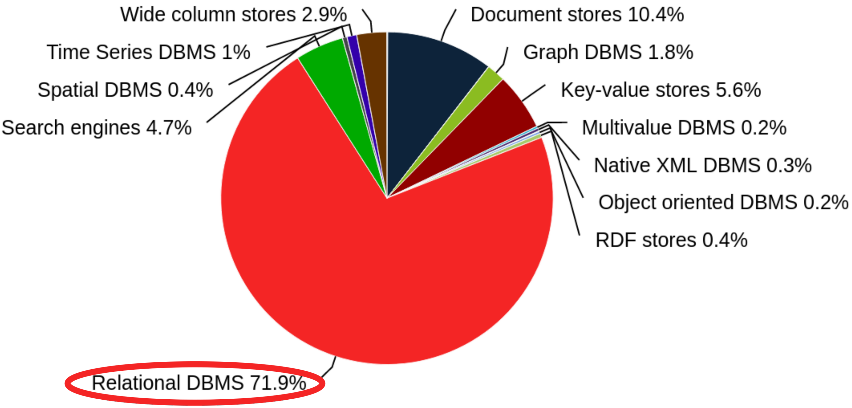
Order table

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

Order-content table

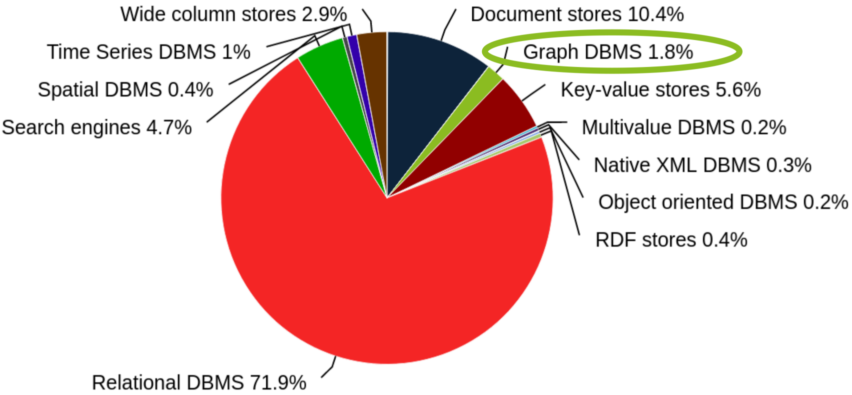
order_id	product
0	Rabbit
0	Pocket Watch
1	Sponge
1	Broom

Vast majority of DMBS are relational, not graph



Figure/data from `db-engines.com`, June 2022

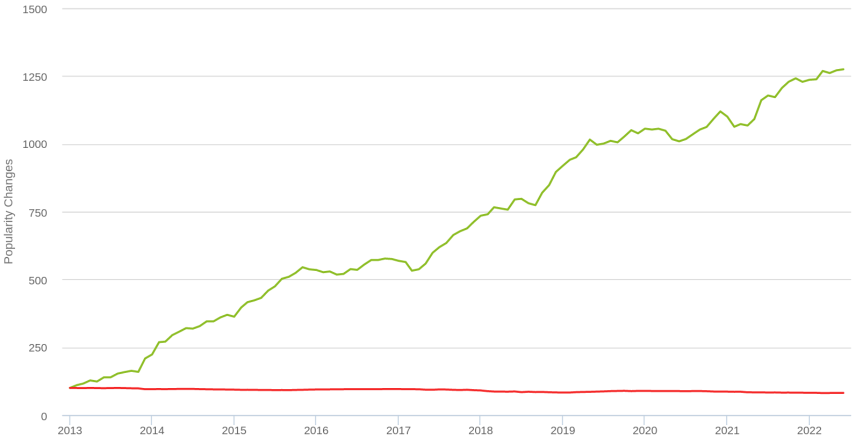
Vast majority of DMBS are relational, not graph



Figure/data from `db-engines.com`, June 2022

Graph DBMS is growing in popularity

+25% per year since 2013



Figure/data from `db-engines.com`, June 2022

Why use graph databases?

Some data have intrinsically the structure of graphs:

- Semantic web
- Network (social, transport, etc.)
- Financial transactions

Why use graph databases?

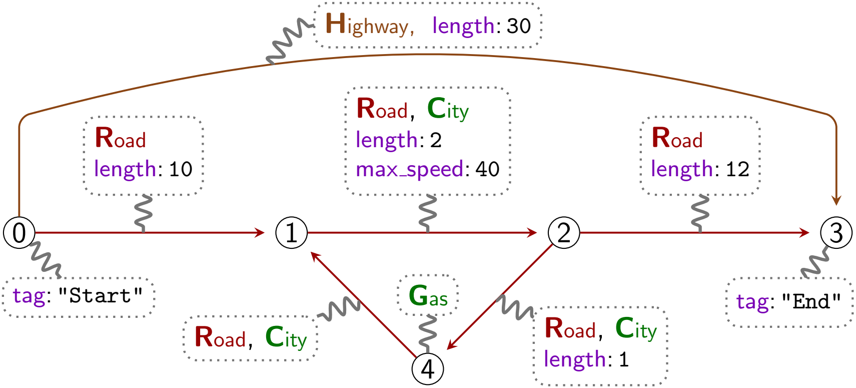
Some data have intrinsically the structure of graphs:

- Semantic web
- Network (social, transport, etc.)
- Financial transactions

Native representation of data as graphs allows:

- Specific algorithms on graphs
- Pattern matching
- More intuitive understanding/visualisation

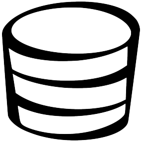
Glimpse at our property graph example



Need for abstraction

Data model
in documentation

DBMS

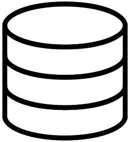


Property graph
(vendor dependent)

Neo4j,
TigerGraph,
etc.

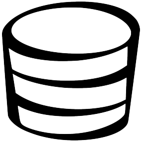
Need for abstraction

Abstract
data model



Abstract
property graph

Data model
in documentation



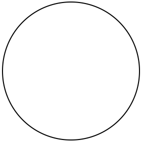
Property graph
(vendor dependent)

DBMS

Neo4j,
TigerGraph,
etc.

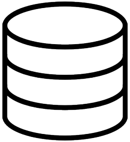
Need for abstraction

Dataless model



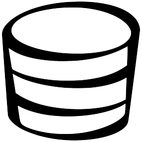
Graph

Abstract
data model



Abstract
property graph

Data model
in documentation



Property graph
(vendor dependent)

DBMS

Neo4j,
TigerGraph,
etc.

Outline

Part 1: without data

- Graphs as database
- Foundation of querying graph databases: RPQs
- Ensuring finitess

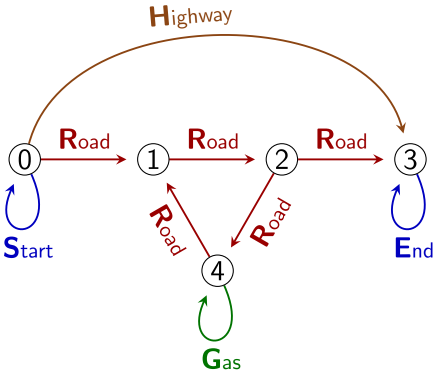
Part 2: with data

- Property graphs
- Cypher
- GQL
- Appendix

Graphs as database

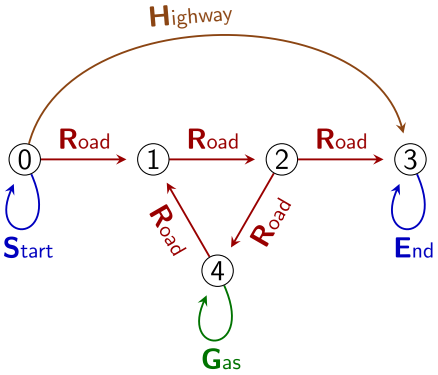
Our example graph

A graph consists of ...



A graph consists of ...

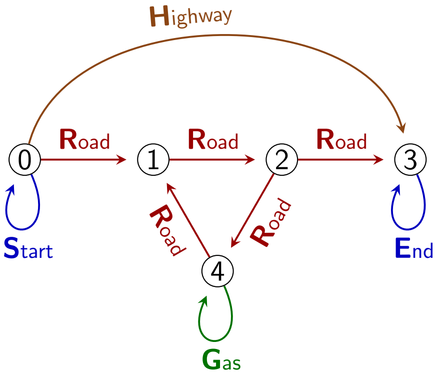
- Vertices (a.k.a. Nodes)
  0, 1, 2, 3, 4



Our example graph

A graph consists of ...

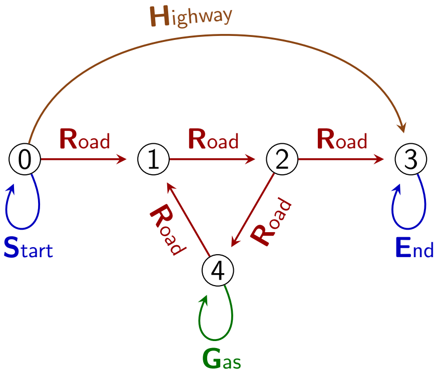
- Vertices (a.k.a. Nodes)
  $0, 1, 2, 3, 4$
- Edges (a.k.a. Relationships)
  $0 \to 0, \quad 0 \to 1, \quad$ etc



Our example graph

A graph consists of ...

- Vertices (a.k.a. Nodes)
  $0, 1, 2, 3, 4$
- Edges (a.k.a. Relationships)
  $0 \to 0, \quad 0 \to 1, \quad$ etc
- Edge labels: $\{$**R**, **H**, **G**, **S**, **E**$\}$



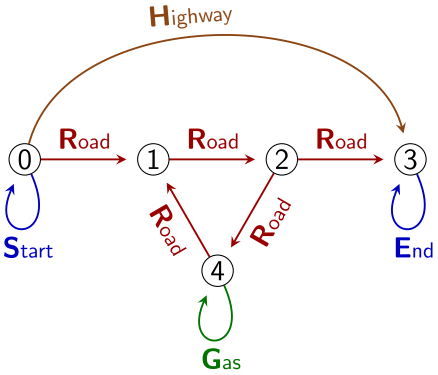
Our example graph

A graph consists of ...

- Vertices (a.k.a. Nodes)
  $0, 1, 2, 3, 4$
- Edges (a.k.a. Relationships)
  $0 \to 0$, $0 \to 1$, etc
- Edge labels: {**R**, **H**, **G**, **S**, **E**}

Walk

- a.k.a. Path
- Sequence of edges
- Can reuse vertices and edges



Our example graph

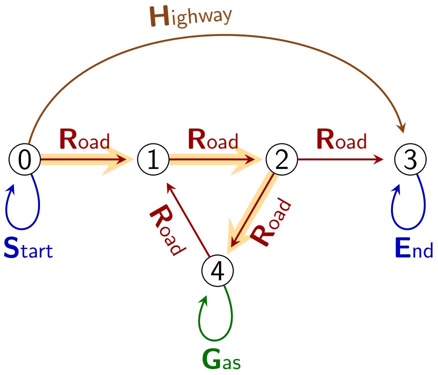
A graph consists of ...

- Vertices (a.k.a. Nodes)
  0, 1, 2, 3, 4
- Edges (a.k.a. Relationships)
  $0 \to 0, \quad 0 \to 1, \quad$ etc
- Edge labels: {**R**, **H**, **G**, **S**, **E**}

Walk

- a.k.a. Path
- Sequence of edges
- Can reuse vertices and edges

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



Our example graph

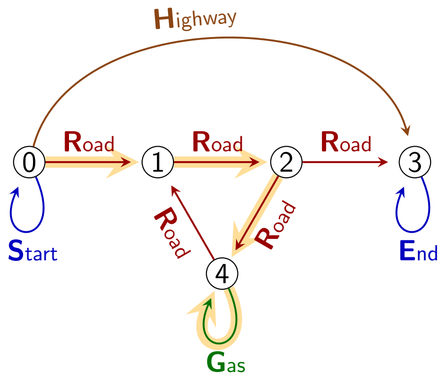
A graph consists of ...

- Vertices (a.k.a. Nodes)
  0, 1, 2, 3, 4
- Edges (a.k.a. Relationships)
  $0 \to 0$, $0 \to 1$, etc
- Edge labels: {**R**, **H**, **G**, **S**, **E**}

Walk

- a.k.a. Path
- Sequence of edges
- Can reuse vertices and edges

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



Our example graph

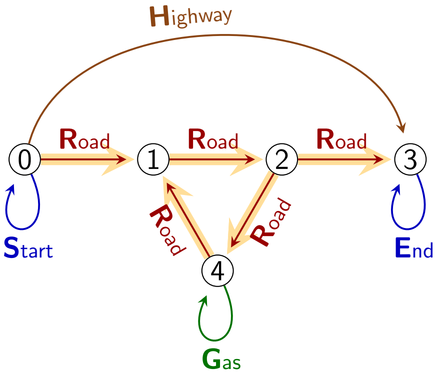
A graph consists of ...

- Vertices (a.k.a. Nodes)
  0, 1, 2, 3, 4
- Edges (a.k.a. Relationships)
  $0 \to 0$, $0 \to 1$, etc
- Edge labels: {**R**, **H**, **G**, **S**, **E**}

Walk

- a.k.a. Path
- Sequence of edges
- Can reuse vertices and edges

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



A graph can be stored in tables

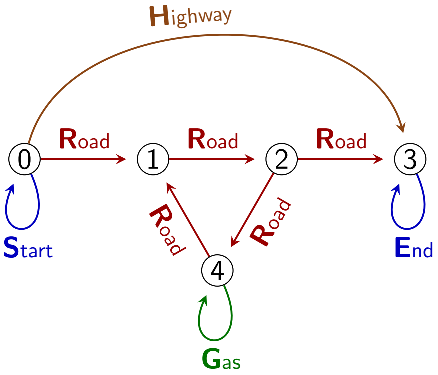
Table for nodes

id
0
1
2
⋮

Table for Road edges

source	target
0	1
1	2
⋮	⋮

One table for Start,
One table for Highway,
etc...



A graph can be stored in tables

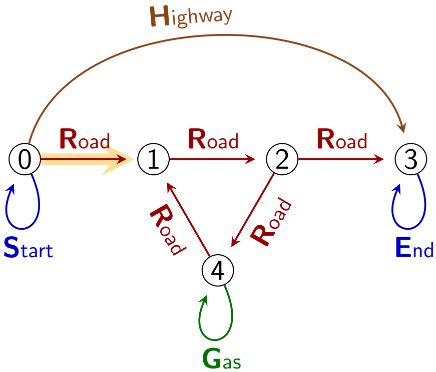
Table for nodes

id
0
1
2
⋮

Table for Road edges

source	target
0	1
1	2
⋮	⋮

One table for Start,
One table for Highway,
etc…

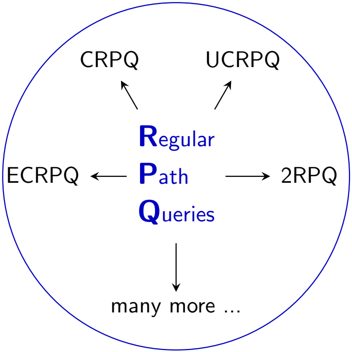


Foundation of querying graph databases: RPQs

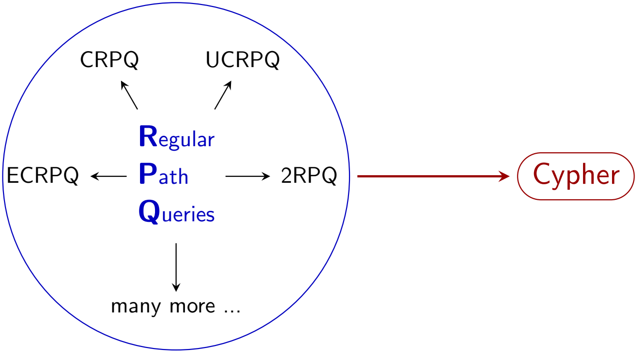
Why RPQs are important

Regular
Path
Queries

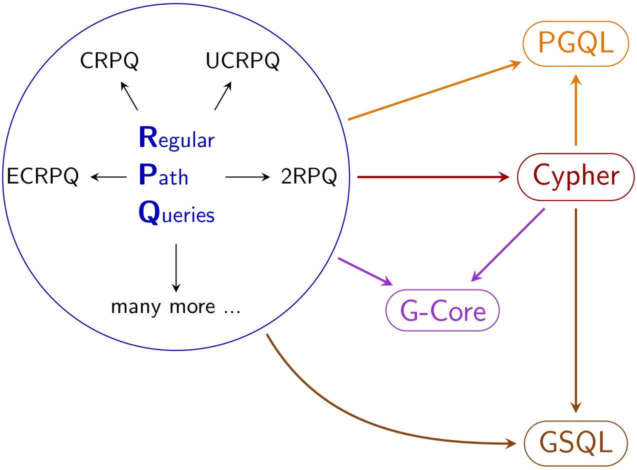
Why RPQs are important



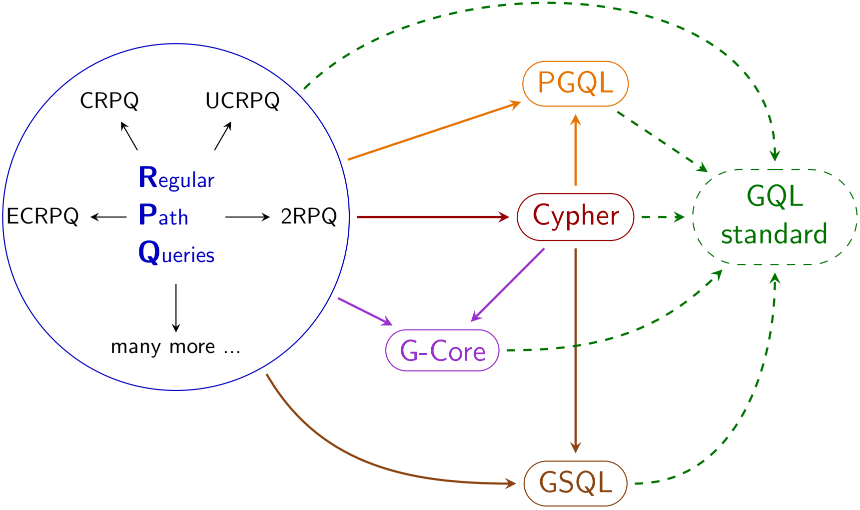
Why RPQs are important



Why RPQs are important



Why RPQs are important



RPQ = Regular expression

$\mathcal{Q} ::=$	**A**	Atoms
	$\mathcal{Q}\mathcal{Q}$	Concatenation
	$\mathcal{Q} + \mathcal{Q}$	Union/Disjunction
	$\mathcal{Q}^*$	Kleene star

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

An RPQ denotes a set of words

S (**H**+**R**)* **E** denotes the words of the shape **S**`<something>`**E**

Any number of **H** and **R**, in any order

RPQ = Regular expression

$\mathcal{Q} ::=$	**A**	Atoms
	$\mathcal{Q}\mathcal{Q}$	Concatenation
	$\mathcal{Q} + \mathcal{Q}$	Union/Disjunction
	$\mathcal{Q}^*$	Kleene star

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

An RPQ denotes a set of words

S (**H**+**R**)* **E** denotes the words of the shape **S**`<something>`**E**

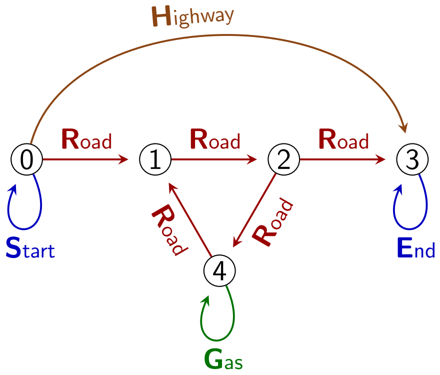
Any number of **H** and **R**, in any order

Matches

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

Query R matches...

...every **R**oad-edge

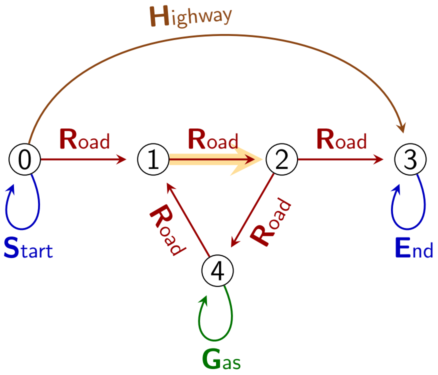


Edge and concatenation

Query R matches...

...every Road-edge

0 → 1	2 → 4
1 → 2	4 → 1
2 → 3	



Edge and concatenation

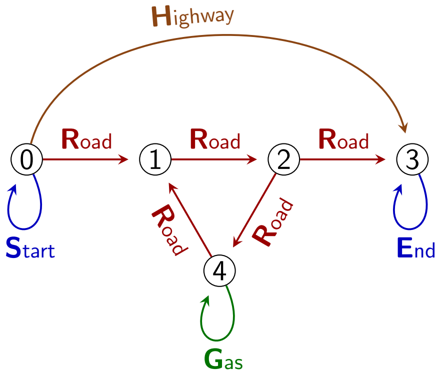
Query **R** matches...

...every **R**oad-edge

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

Query **RR** matches...

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



Edge and concatenation

Query R matches...

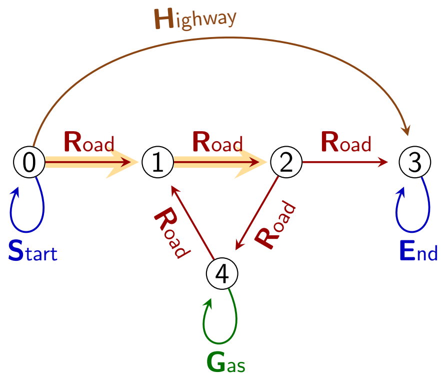
...every Road-edge

0 → 1	2 → 4
1 → 2	4 → 1
2 → 3	

Query RR matches...

...walks of two Road-edges

0 → 1 → 2	2 → 4 → 1
1 → 2 → 3	4 → 1 → 2
1 → 2 → 3	



Edge and concatenation

Query R matches...

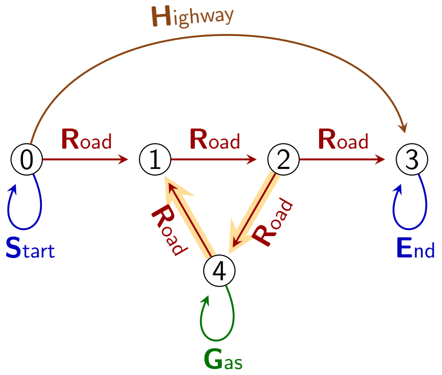
...every Road-edge

0 → 1	2 → 4
1 → 2	4 → 1
2 → 3	

Query RR matches...

...walks of two Road-edges

0 → 1 → 2	2 → 4 → 1
1 → 2 → 3	4 → 1 → 2
1 → 2 → 3	

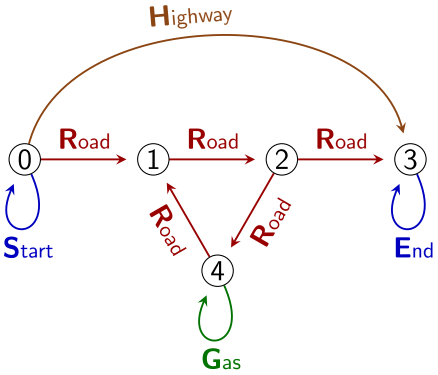


Disjunction

Query R+H matches...

...edges labelled by **R** or by **H**

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

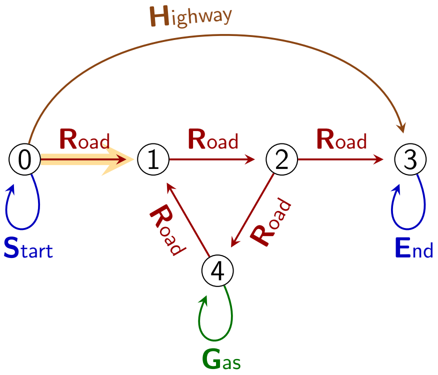


Disjunction

Query **R**+**H** matches...

...edges labelled by **R** or by **H**

0 → 1	2 → 4
1 → 2	4 → 1
2 → 3	0 → 3

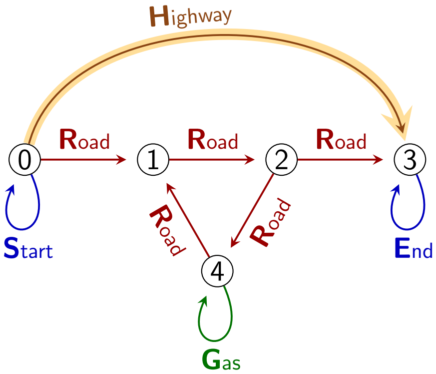


Disjunction

Query R+H matches...

...edges labelled by **R** or by **H**

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



Disjunction

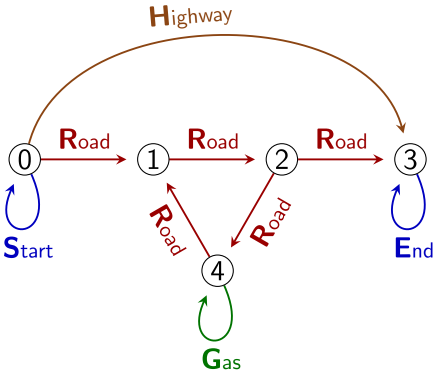
Query **R** + **H** matches…

…edges labelled by **R** or by **H**

0 → 1	2 → 4
1 → 2	4 → 1
2 → 3	0 → 3

Query **S** (**H** + **R**) matches

0 → 0 → 1
0 → 0 → 3

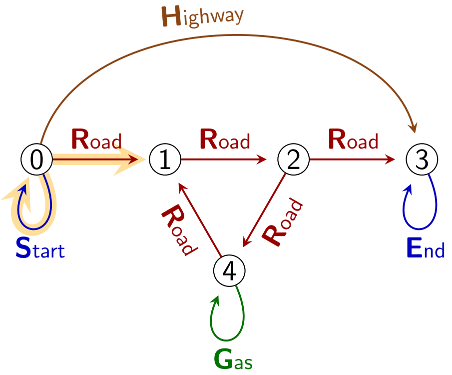


Disjunction

Query R + H matches...

...edges labelled by **R** or by **H**

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



Query S (H + R) matches

$0 \to 0 \to 1$

$0 \to 0 \to 3$

Disjunction

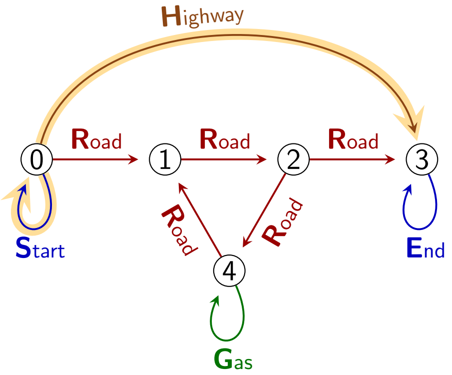
Query R+H matches...

...edges labelled by **R** or by **H**

0 → 1	2 → 4
1 → 2	4 → 1
2 → 3	0 → 3

Query S (H+R) matches

0 → 0 → 1
0 → 0 → 3



Kleene star

Query **R*** matches...

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

$0 \to 1$

$0 \to 1 \to 2$

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

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

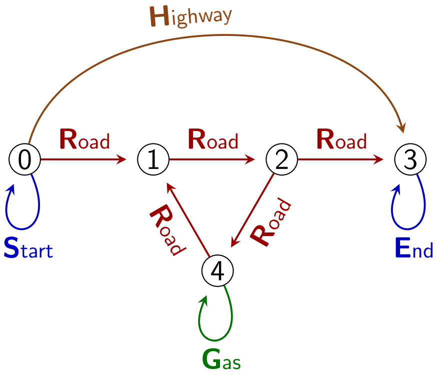
$\vdots$

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

$\vdots$

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

$\vdots$



Kleene star

Query $\mathbf{R}^*$ matches...

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

$0 \rightarrow 1$

$0 \rightarrow 1 \rightarrow 2$

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3$

$0 \rightarrow 1 \rightarrow 2 \rightarrow 4$

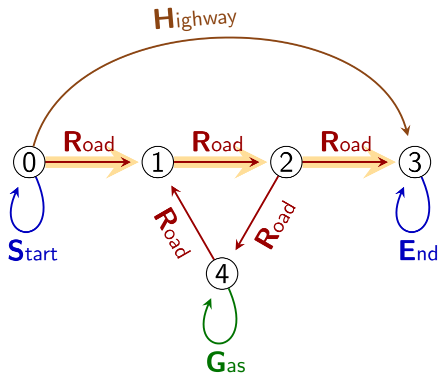
$\vdots$

$1 \rightarrow 2 \rightarrow 4 \rightarrow 1$

$\vdots$

$1 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 1$

$\vdots$



Query **R*** matches...

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

$0 \to 1$

$0 \to 1 \to 2$

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

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

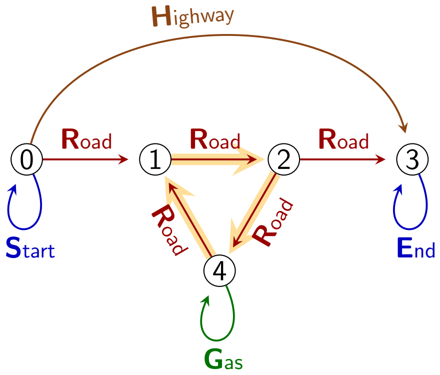
$\vdots$

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

$\vdots$

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

$\vdots$



Kleene star

Query **R*** matches...

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

$0 \to 1$

$0 \to 1 \to 2$

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

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

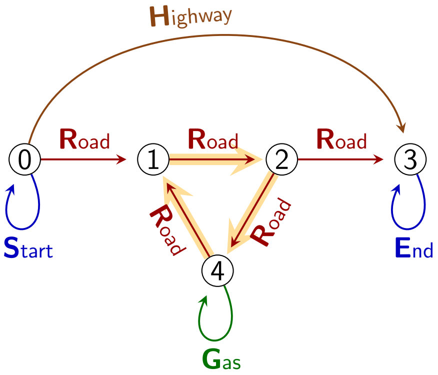
$\vdots$

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

$\vdots$

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

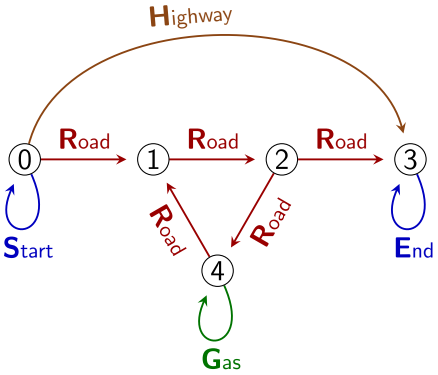
$\vdots$



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

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

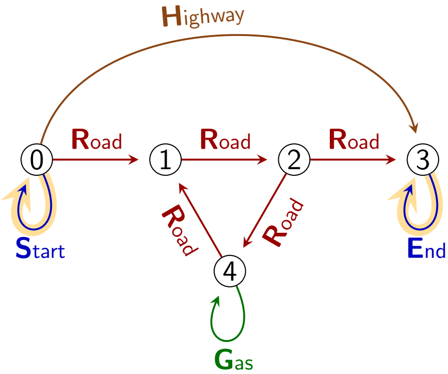
$\mathcal{Q}_1$ matches...



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

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

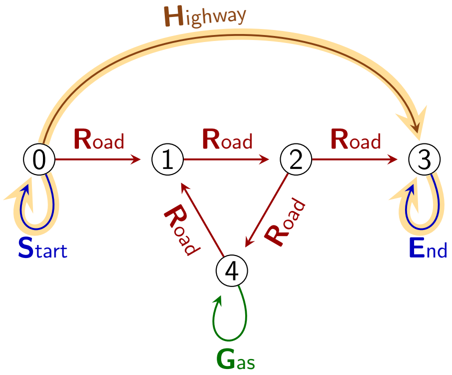
$\mathcal{Q}_1$ matches...



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

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

- The highway

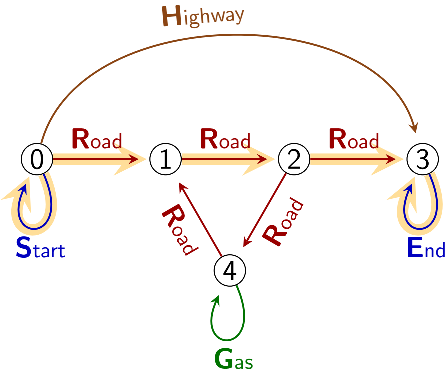


The queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$

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

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

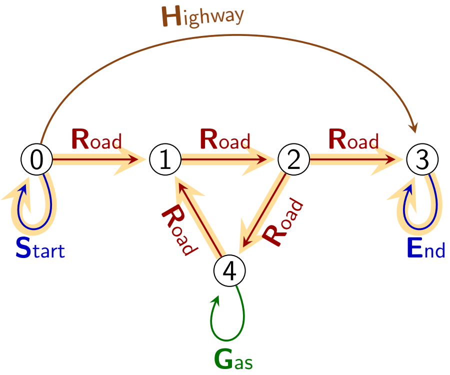
- The highway
- The direct road



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

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

- The highway
- The direct road
- Road with laps in the circuit



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

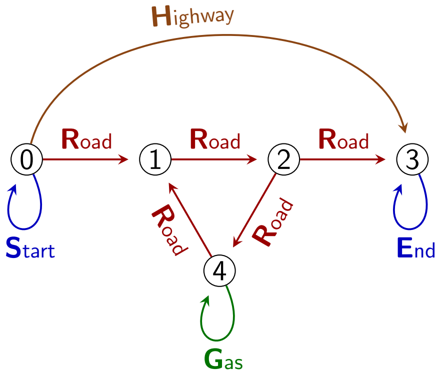
$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{H})^*\,\mathbf{E}$

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

- The highway
- The direct road
- Road with laps in the circuit

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

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



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

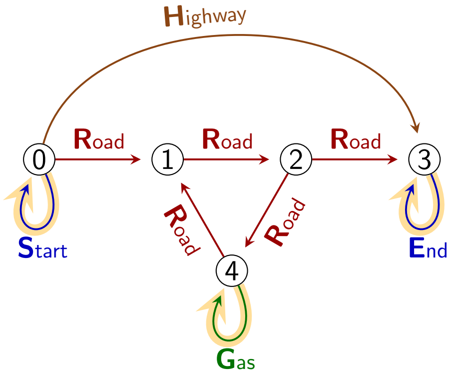
$\mathcal{Q}_1 = \mathbf{S}\,(\mathbf{R}+\mathbf{H})^*\,\mathbf{E}$

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

- The highway
- The direct road
- Road with laps in the circuit

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

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



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

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

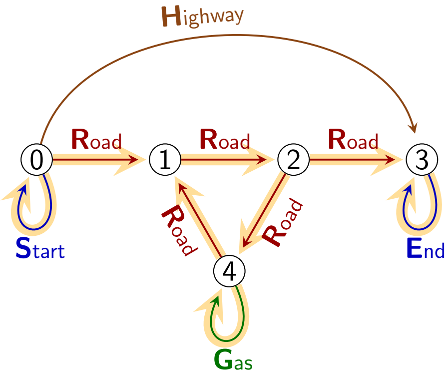
$\mathcal{Q}_1$ matches...

- The highway
- The direct road
- Road with laps in the circuit

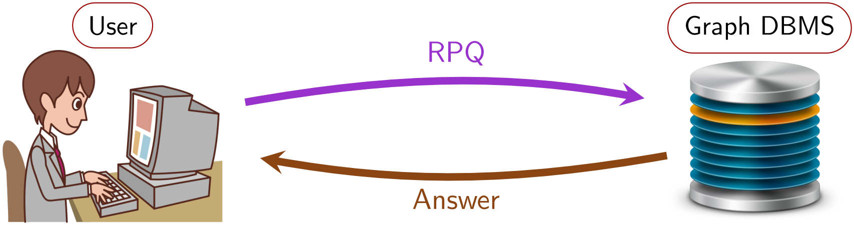
$\mathcal{Q}_2 = \mathbf{S}\,(\mathbf{R}+\mathbf{H})^*\,\mathbf{G}\,(\mathbf{R}+\mathbf{H})^*\,\mathbf{E}$

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

- Road with laps in the circuit



Fundamental challenge with RPQs



⚠ Answer may be infinite ⚠

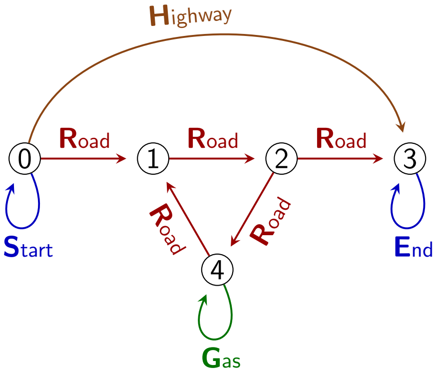
Ensuring finitess

Homomorphism semantics

Main theoretical semantics [Angles et al. 2017]

Definition

- Returns the endpoints of matches



Homomorphism semantics

Main theoretical semantics [Angles et al. 2017]

Definition

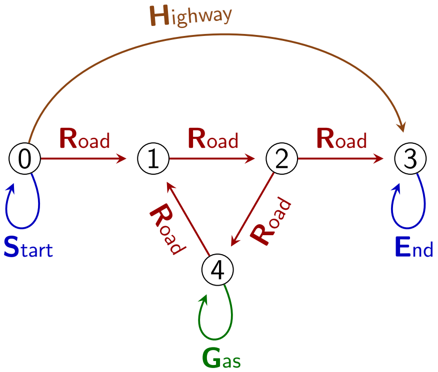
- Returns the endpoints of matches

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

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

- All matches are: $0 \to \cdots \to 3$

$\Rightarrow \mathcal{Q}_1$ 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

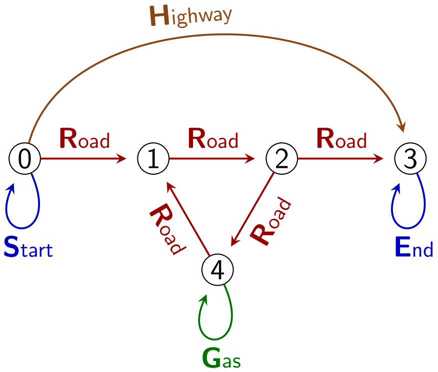
- Limited information in the answer
  - User: *"I want to go from Paris to Lyon by car"*
  - Database: *"Yes you can"*

Shortest-walk semantics

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

Definition

- Return the walk with the least number of edges



Shortest-walk semantics

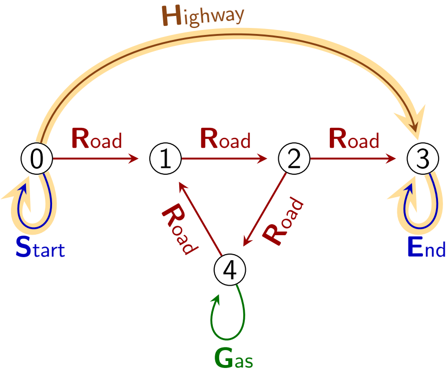
PGQL (Oracle), GSQL (TigerGraph), G-core [Angles et al. 2018]

Definition

- Return the walk with the least number of edges

$\mathcal{Q}_1 =$ **S** (**R**+**H**)* **E**

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



Shortest-walk semantics

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

Definition

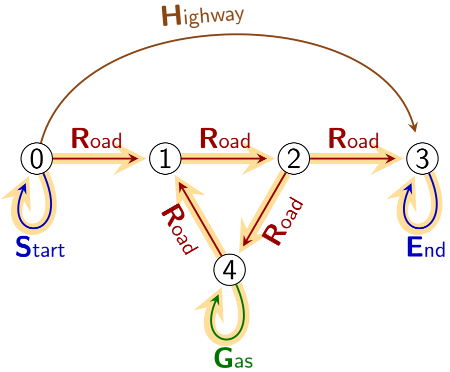
- Return the walk with the least number of edges

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

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

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

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



Shortest walk semantics (2)

Pros and cons

Pros

- Returns walks
- Efficient algorithms
- Horizontal post-processing
  - Horizontal = along the walk
  - *"Is there a gas station on the way?"*
  - *"What is the length of the walk?"*

Shortest walk semantics (2)

Pros and cons

Pros

- Returns walks
- Efficient algorithms
- Horizontal post-processing
  - Horizontal = along the walk
  - *"Is there a gas station on the way?"*
  - *"What is the length of the walk?"*

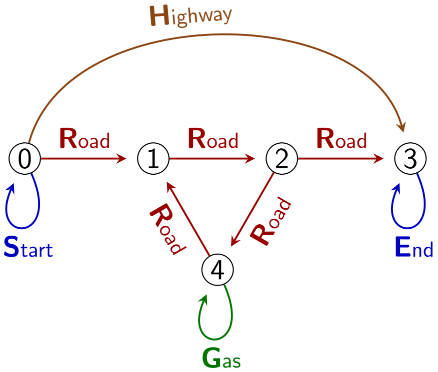
Cons

- No vertical post-processing
  - Vertical = accross the walks with the same endpoints
  - *"What is the shortest walk in time?"*
  - *"What is the connectedness level?"*

Cypher (Neo4j)

Definition

- Return walks
- Forbidden to repeat edges

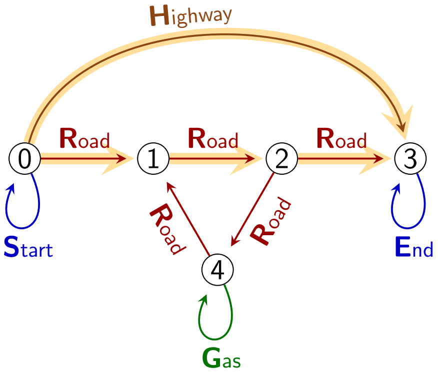


Definition

- Return walks
- Forbidden to repeat edges

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

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



Trail semantics

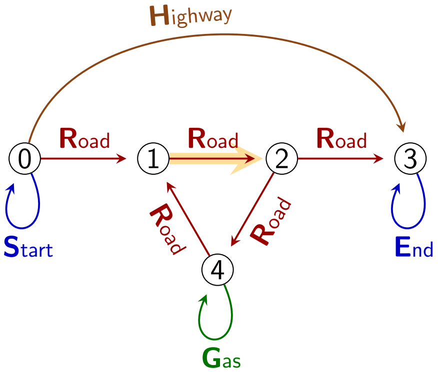
Cypher (Neo4j)

Definition

- Return walks
- Forbidden to repeat edges

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

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



Definition

- Return walks
- Forbidden to repeat edges

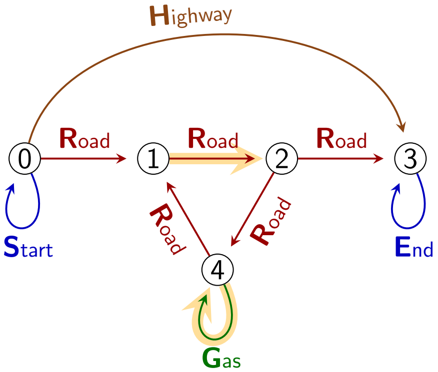
$\mathcal{Q}_1 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{H})^*\ \mathbf{E}$

- $\mathcal{Q}_1$ returns 2 walks
  - the highway
  - the straight road
- Walks with circuit laps

  $\Rightarrow$ repeat the middle edge

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

- $\mathcal{Q}_2$ returns nothing



Trail semantics (2)

Pros and cons

Pros

- Returns walks
- Counting matches
- Horizontal and vertical post-processing

Trail semantics (2)

Pros and cons

Pros

- Returns walks
- Counting matches
- Horizontal and vertical post-processing

Cons

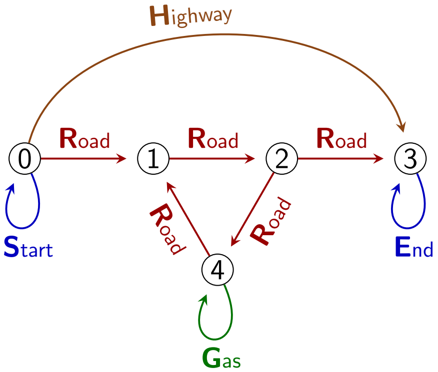
- All problems are computationally hard [Martens et al. 2020]
  - Counting, enumeration, existence
  - Checking whteher Q_2 returns anything $\rightarrow$ Already hard
- No repeated edge $\rightarrow$ not always intuitive

Run-based semantics

New theoretical proposal [David-Francis-Marsault 2023?]

Definition

- Returns walks
- Each edge may match each atom only once



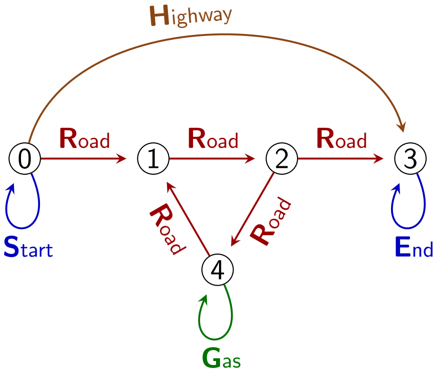
Run-based semantics

New theoretical proposal [David-Francis-Marsault 2023?]

Definition

- Returns walks
- Each edge may match each atom only once

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



Run-based semantics

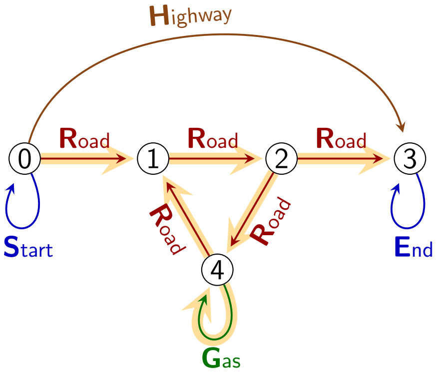
New theoretical proposal [David-Francis-Marsault 2023?]

Definition

- Returns walks
- Each edge may match each atom only once

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

- Returns the 1-lap road only



Run-based semantics

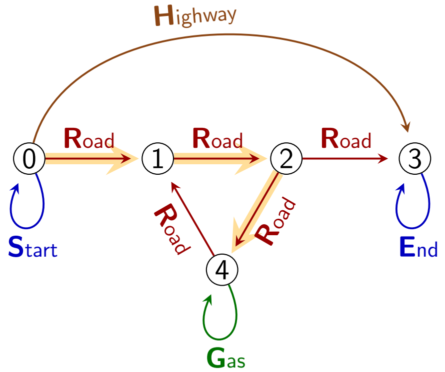
New theoretical proposal [David-Francis-Marsault 2023?]

Definition

- Returns walks
- Each edge may match each atom only once

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

- Returns the 1-lap road only
  - Before **G** $\rightarrow$ use the left **R**



Run-based semantics

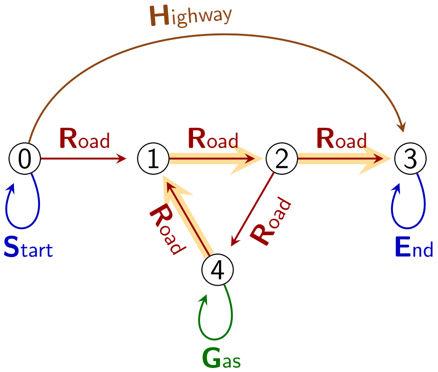
New theoretical proposal [David-Francis-Marsault 2023?]

Definition

- Returns walks
- Each edge may match each atom only once

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

- Returns the 1-lap road only
  - Before **G** $\rightarrow$ use the left **R**
  - After **G** $\rightarrow$ use the right **R**



Run-based semantics

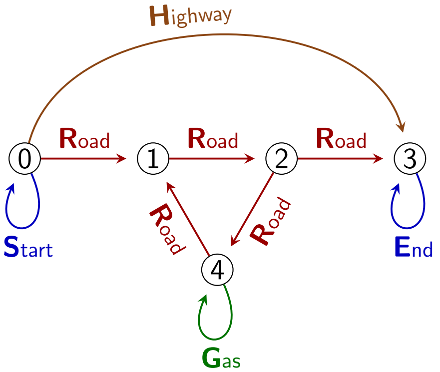
New theoretical proposal [David-Francis-Marsault 2023?]

Definition

- Returns walks
- Each edge may match each atom only once

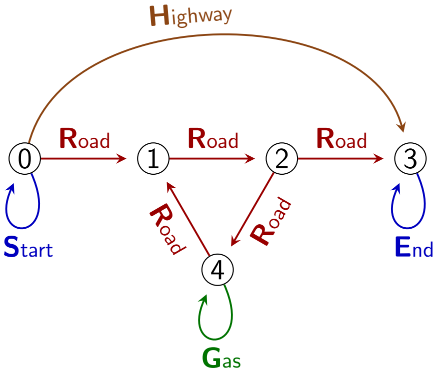
$\mathcal{Q}_2 = \ \mathbf{S}\ (\mathbf{R}+\mathbf{H})^*\ \mathbf{G}\ (\mathbf{R}+\mathbf{H})^*\ \mathbf{E}$

- Returns the 1-lap road only
  - Before **G** $\rightarrow$ use the left **R**
  - After **G** $\rightarrow$ use the right **R**
- > 1 circuit lap $\Rightarrow$ some edge use the same atom twice



Definition

- Returns walks
- Each edge may match each atom only once



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

- Returns the 1-lap road only
  - Before **G** $\rightarrow$ use the left **R**
  - After **G** $\rightarrow$ use the right **R**
- > 1 circuit lap $\Rightarrow$ some edge use the same atom twice

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

- Returns the highway and the straight road

Run-based semantics (2)

Pros and cons

Pros

- Returns walks
- Horizontal and vertical post-processing
- Enumerating matches is efficient
- Counting matches

Run-based semantics (2)

Pros and cons

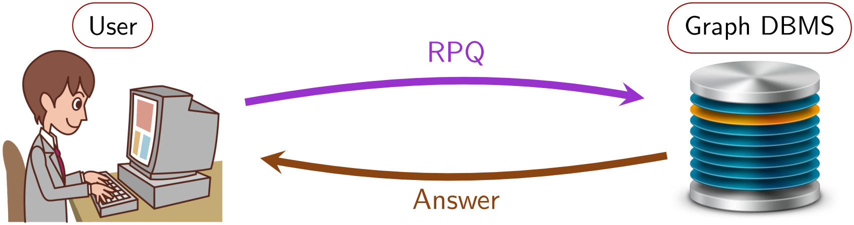
Pros

- Returns walks
- Horizontal and vertical post-processing
- Enumerating matches is efficient
- Counting matches

Cons

- Counting matches is computationally hard
- Answer depends on the way the query is written
  - $\mathbf{R}^*$ allows no lap in the circuit
  - $\mathbf{R}^*\mathbf{R}^*$ allows 1 lap in the circuit
  - $\mathbf{R}^*\mathbf{R}^*\mathbf{R}^*$ allows 2 lap in the circuit
  - etc.

Fundamental challenge with RPQs

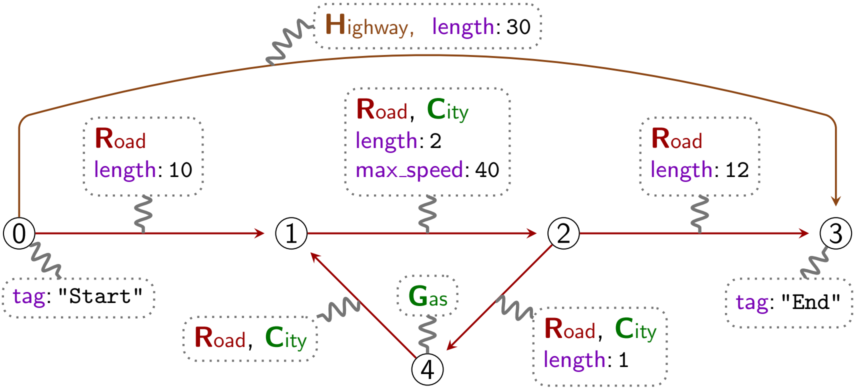


⚠ Answer may be infinite ⚠

- Several way to ensure finiteness
  - Homomorphism semantics → Returns litte information
  - Shortest-walk semantics → No vertical post-processing
  - Trail semantics → Not efficient in bad cases
  - etc.
- No solution is clearly superior

Property graphs

Example property graph

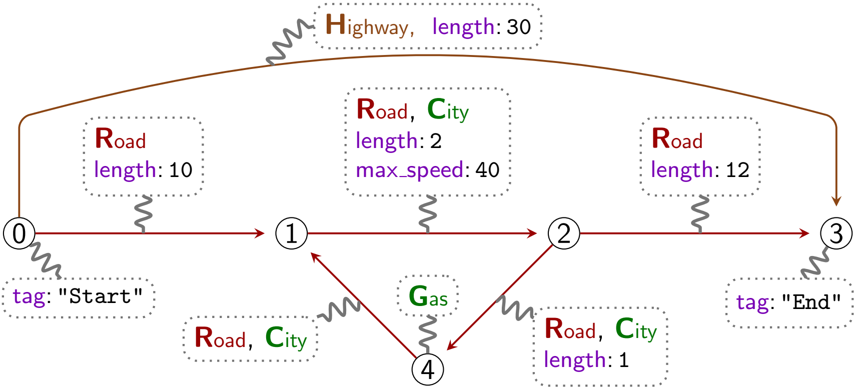


Vertices and edges may bear:

- zero or more labels
- zero or more properties

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

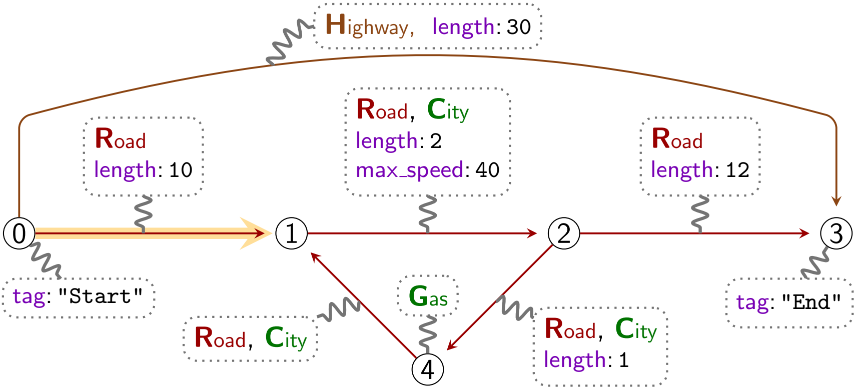
Storing our property graph into tables



id	source_id	target_id	Road	Highway	City	length	max_speed
e_{01}	0	1	true	false	false	10	
e_{12}	1	2	true	false	true	10	40
e_{41}	4	1	true	false	true		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Graph | Labels | Properties

Storing our property graph into tables



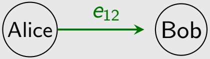
	id	source_id	target_id	Road	Highway	City	length	max_speed	
▶	e_{01}	0	1	true	false	false	10		◀
	e_{12}	1	2	true	false	true	10	40	
	e_{41}	4	1	true	false	true			
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Graph — Labels — Properties

What property graphs can't do

Wedding is edgy...

id	person1	person2
e_{12}	Alice	Bob
⋮	⋮	⋮

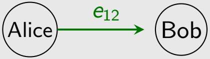


..but trouple is trouble

What property graphs can't do

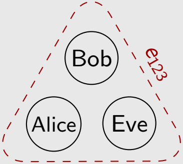
Wedding is edgy...

id	person1	person2
e_{12}	Alice	Bob
⋮	⋮	⋮



..but trouple is trouble

id	person1	person2	person3
e_{123}	Alice	Bob	Eve
⋮	⋮	⋮	⋮



Cypher

History

- Created for Neo4j DBMS in 2011
- Neo4j and Cypher become popular
- Project openCypher since 2015
  - Goal: standardize Cypher
  - Evolution led by community (oCIM)
  - Formal semantics [Francis et al. 2018] [Green et al. 2019]
  - Replaced by GQL
- Implemented in SAP HANA, Redis graph, etc.

ASCII-art syntax for RPQs

- Vertices: `MATCH (:Gas)`

ASCII-art syntax for RPQs

- Vertices: `MATCH (:Gas)`
- Edges: `MATCH -[:Road]->`

ASCII-art syntax for RPQs

- Vertices: `MATCH (:Gas)`
- Edges: `MATCH -[:Road]->`
- Disjunction: `MATCH -[:Road|Highway]->`

ASCII-art syntax for RPQs

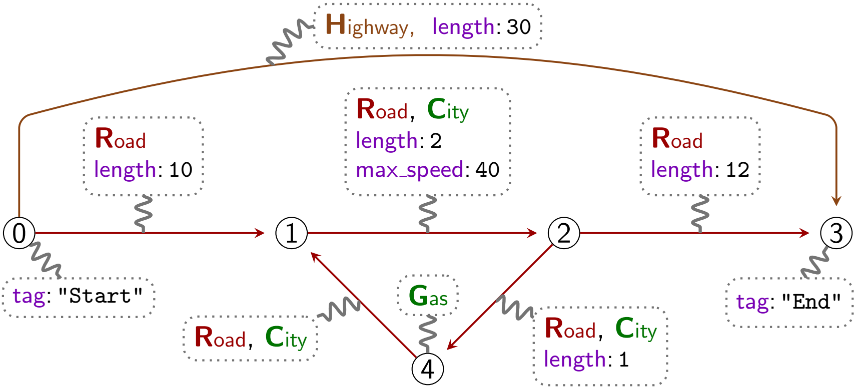
- Vertices: `MATCH (:Gas)`
- Edges: `MATCH -[:Road]->`
- Disjunction: `MATCH -[:Road|Highway]->`
- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`

ASCII-art syntax for RPQs

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

- Vertices: `MATCH (:Gas)`
- Edges: `MATCH -[:Road]->`
- Disjunction: `MATCH -[:Road|Highway]->`
- Concatenation: `MATCH ()-[:Road]->(:Gas)-[:Road]->()`
- Kleene star: `MATCH ()-[:Road*]->()`
- $\mathcal{Q}_1$:
  `MATCH ({tag:"Start"})-[:Road|Highway*]->({tag:"End"})`

Cypher returns a table...



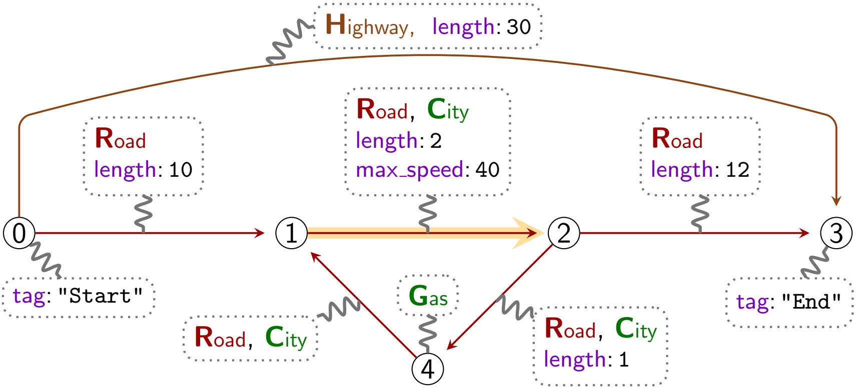
Query

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

Result

s	t
1	2
2	4
4	1

Cypher returns a table...



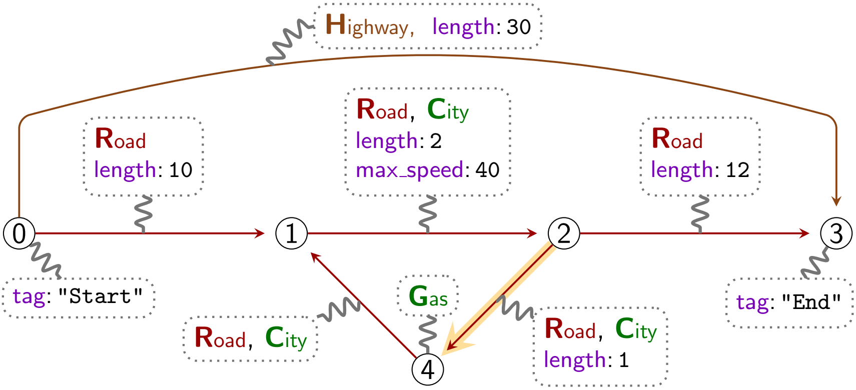
Query

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

Result

s	t
1	2
2	4
4	1

Cypher returns a table...



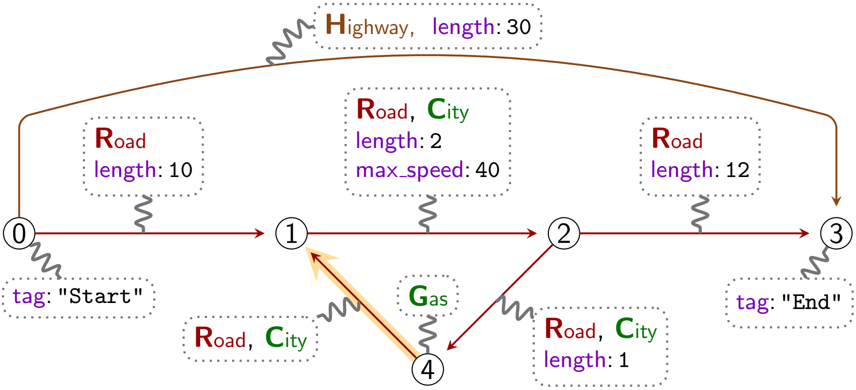
Query

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

Result

s	t
1	2
2	4
4	1

Cypher returns a table...

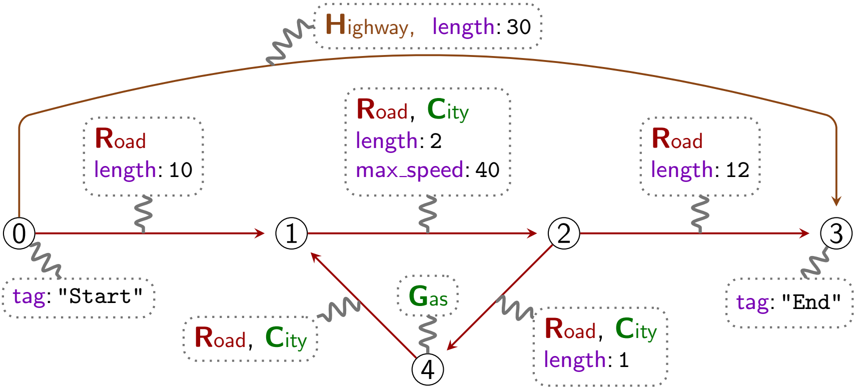


Query

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

Result

s	t
1	2
2	4
4	1



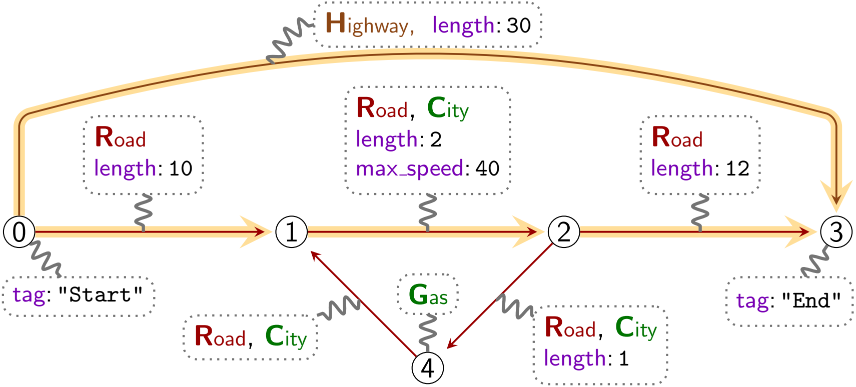
Query

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

Result

s	t
0	3
0	3

Cypher returns a table… but computes walks



Query

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

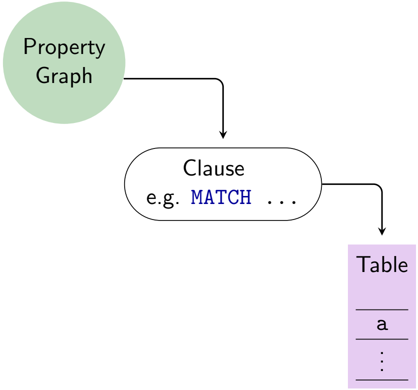
Result

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

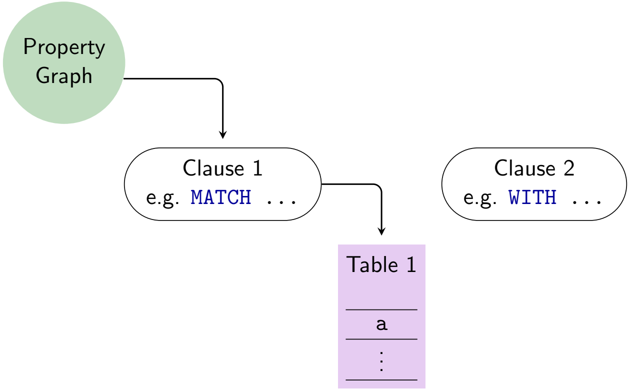
Other Cypher features

- `ORDER BY`: orders row
- `WHERE`: filters row
- `WITH` or `RETURN`:
  - adds/renames columns
  - horizontal aggregation: e.g. `reduce`
  - vertical aggregation: e.g. `count`
- `CREATE`/`DELETE`/`SET`: updates the property graph

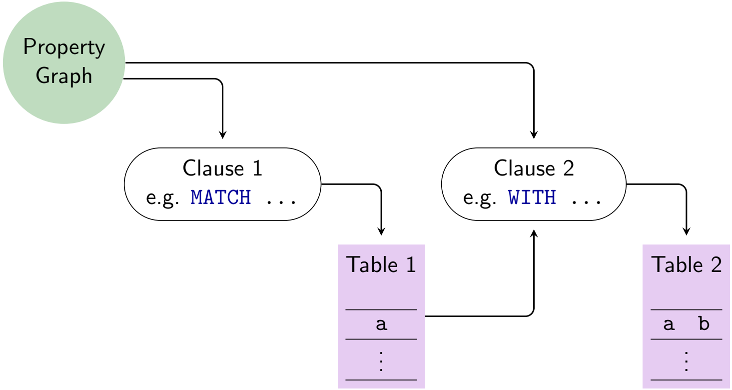
A Cypher query actually chain clauses



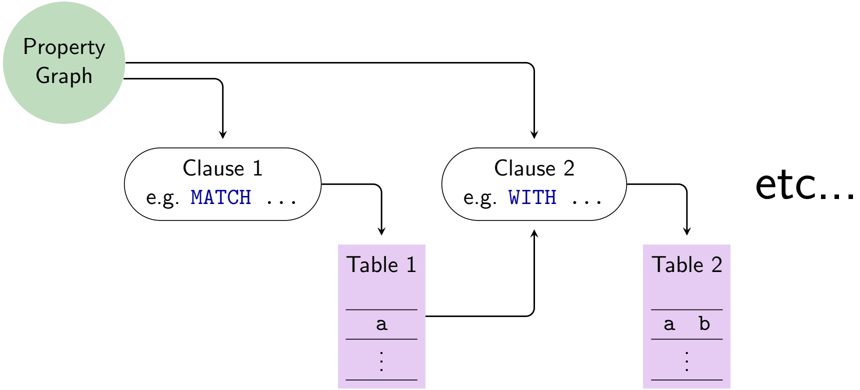
A Cypher query actually chain clauses



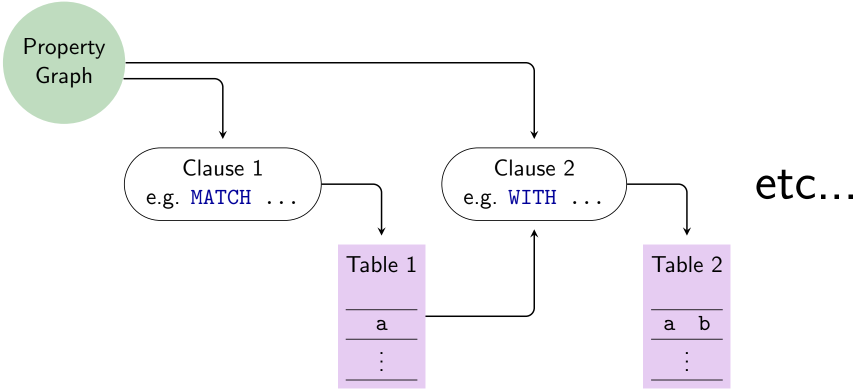
A Cypher query actually chain clauses



A Cypher query actually chain clauses



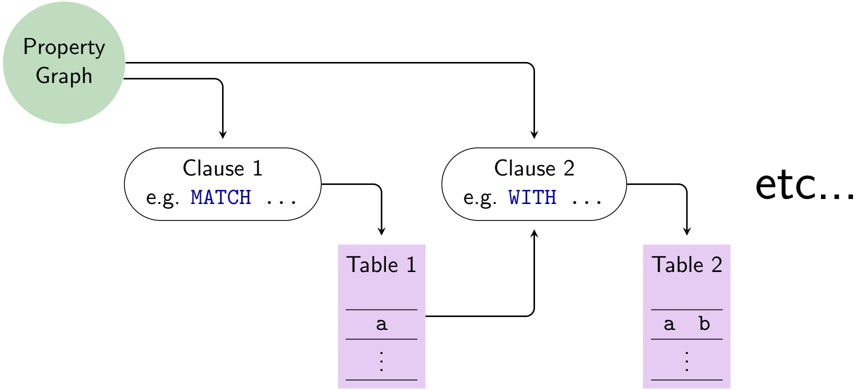
A Cypher query actually chain clauses



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

Since 2015: openCypher aims at making Cypher a standard

GQL History

Since 2015: openCypher aims at making Cypher a standard

2017:

- Cypher won't become a standard !
- Merge existing languages instead ?

GQL History

Since 2015: openCypher aims at making Cypher a standard

2017:

- Cypher won't become a standard !
- Merge existing languages instead ?

2018: Comparison of features in Cypher, G-Core, PGQL

GQL History

Since 2015: openCypher aims at making Cypher a standard

2017:
- Cypher won't become a standard !
- Merge existing languages instead ?

2018: Comparison of features in Cypher, G-Core, PGQL

2019: ISO project [ISO/IEC CD 39075 and 9075-16.2]
- Academia
- Neo4j (Cypher)
- Oracle (SQL, PGQL)
- TigerGraph (GSQL)
- ...

GQL History

Since 2015: openCypher aims at making Cypher a standard

2017:

- Cypher won't become a standard !
- Merge existing languages instead ?

2018: Comparison of features in Cypher, G-Core, PGQL

2019: ISO project [ISO/IEC CD 39075 and 9075-16.2]

- Academia
- Neo4j (Cypher)
- Oracle (SQL, PGQL)
- TigerGraph (GSQL)
- ...

2021: Pattern matching is decided [Alin et al. 2022]

GQL History

Since 2015: openCypher aims at making Cypher a standard

2017:

- Cypher won't become a standard !
- Merge existing languages instead ?

2018: Comparison of features in Cypher, G-Core, PGQL

2019: ISO project [ISO/IEC CD 39075 and 9075-16.2]

- Academia
- Neo4j (Cypher)
- Oracle (SQL, PGQL)
- TigerGraph (GSQL)
- ...

2021: Pattern matching is decided [Alin et al. 2022]

2023 (expected): Publication of version 1 of GQL Standard

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

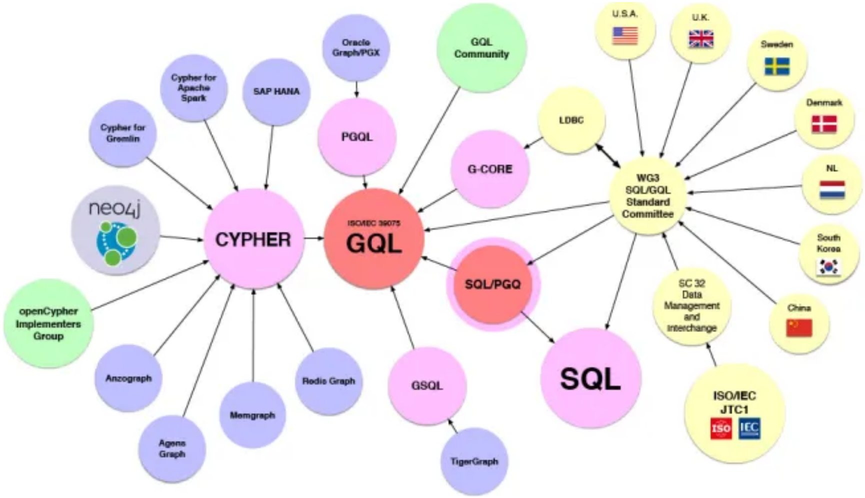
GQL and the fundamental challenge with RPQs

An RPQ may have infinitely many matches

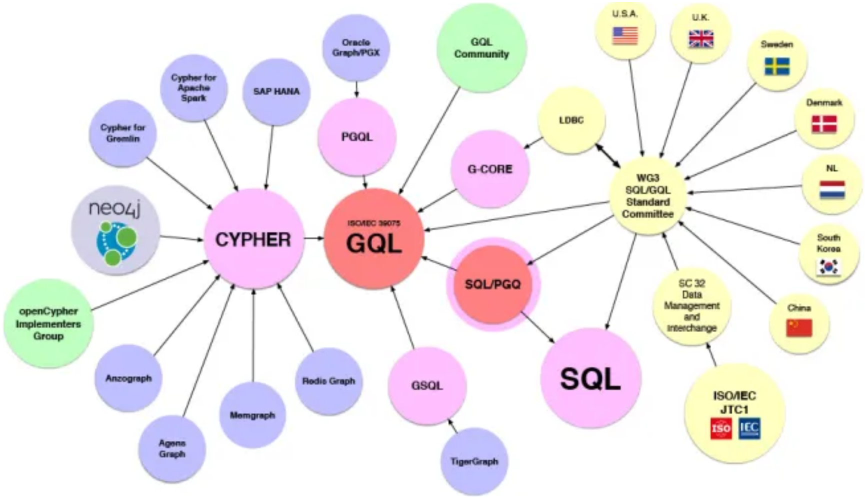
- GQL has to ensure finiteness of answer
- No solution is clearly superior

GQL does not choose

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



Oracle Graph/PGX
GQL Community
U.S.A.
U.K.
Sweden
Denmark
NL
South Korea
China
Cypher for Apache Spark
SAP HANA
PGQL
Cypher for Gremlin
LDBC
G-CORE
WG3 SQL/GQL Standard Committee
neo4j
CYPHER
ISO/IEC 39075
GQL
SQL/PGQ
openCypher Implementers Group
SC 32 Data Management and Interchange
Anzograph
Redis Graph
GSQL
SQL
Memgraph
Agens Graph
TigerGraph
ISO/IEC JTC1
ISO
IEC

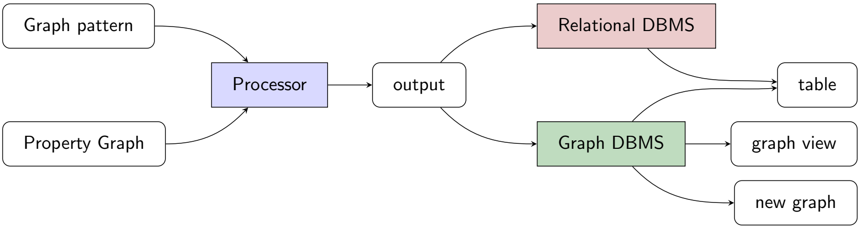


Thank you for your attention !

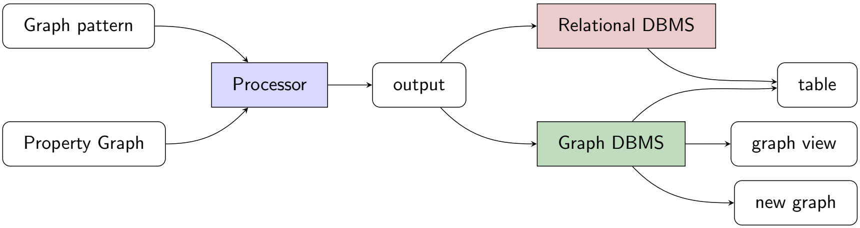
Navigable outline

- Introduction
  - General setting . . . 1
  - Relational DBMS . . . 2
  - Graph DBMS in practice . . . 3
  - Abstraction . . . 7
- Graphs as database
  - Definition by example . . . 9
  - Graph vs Tables . . . 10
- Foundation of querying graph databases: RPQs
  - RPQ = Regular expression . . . 12
  - Examples . . . 13
  - Main queries . . . 16
  - The challenge with RPQs . . . 17
- Ensuring finitess
  - Homomorphism semantics . . . 18
  - Shortest walk . . . 20
  - Trail semantics . . . 22
  - Run-based semantics . . . 24
- Property graphs
  - Definition by example . . . 27
  - Property Graph vs Tables . . . 28
- Cypher
  - History . . . 30
  - Syntax . . . 31
  - Examples . . . 32
- GQL
  - History . . . 40
  - GQL semantics . . . 36
- Appendix

GQL will be usable from SQL

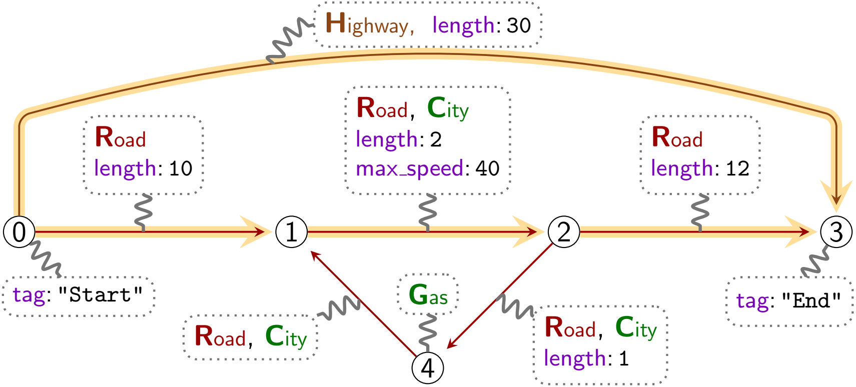


GQL will be usable from SQL



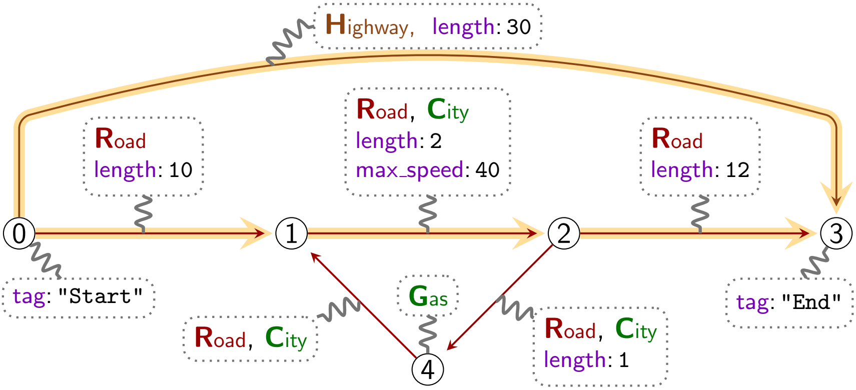
Output of GQL: set of path bindings
Path binding = walk annotated with variables

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	