Recent advances in Graph Data Management

ISWC 2024

Domagoj Vrgoč

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Instituto Milenio Fundamentos de los datos

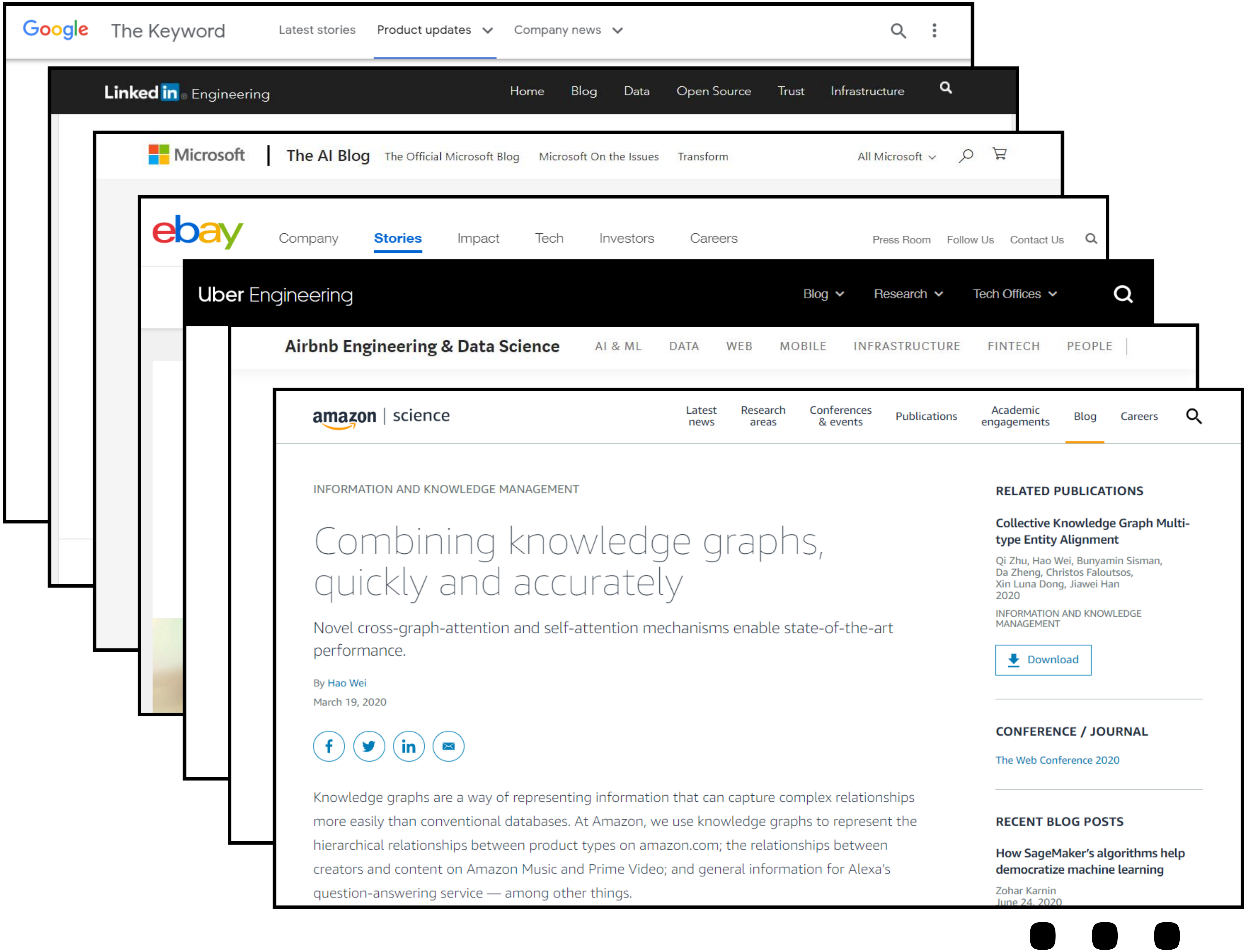
Outline

This is about Graph Databases

- Part 1: Modelling, data and queries
- Part 2: Worst-case optimal join algorithms
- Part 3: Path queries
- Part 4: MillenniumDB

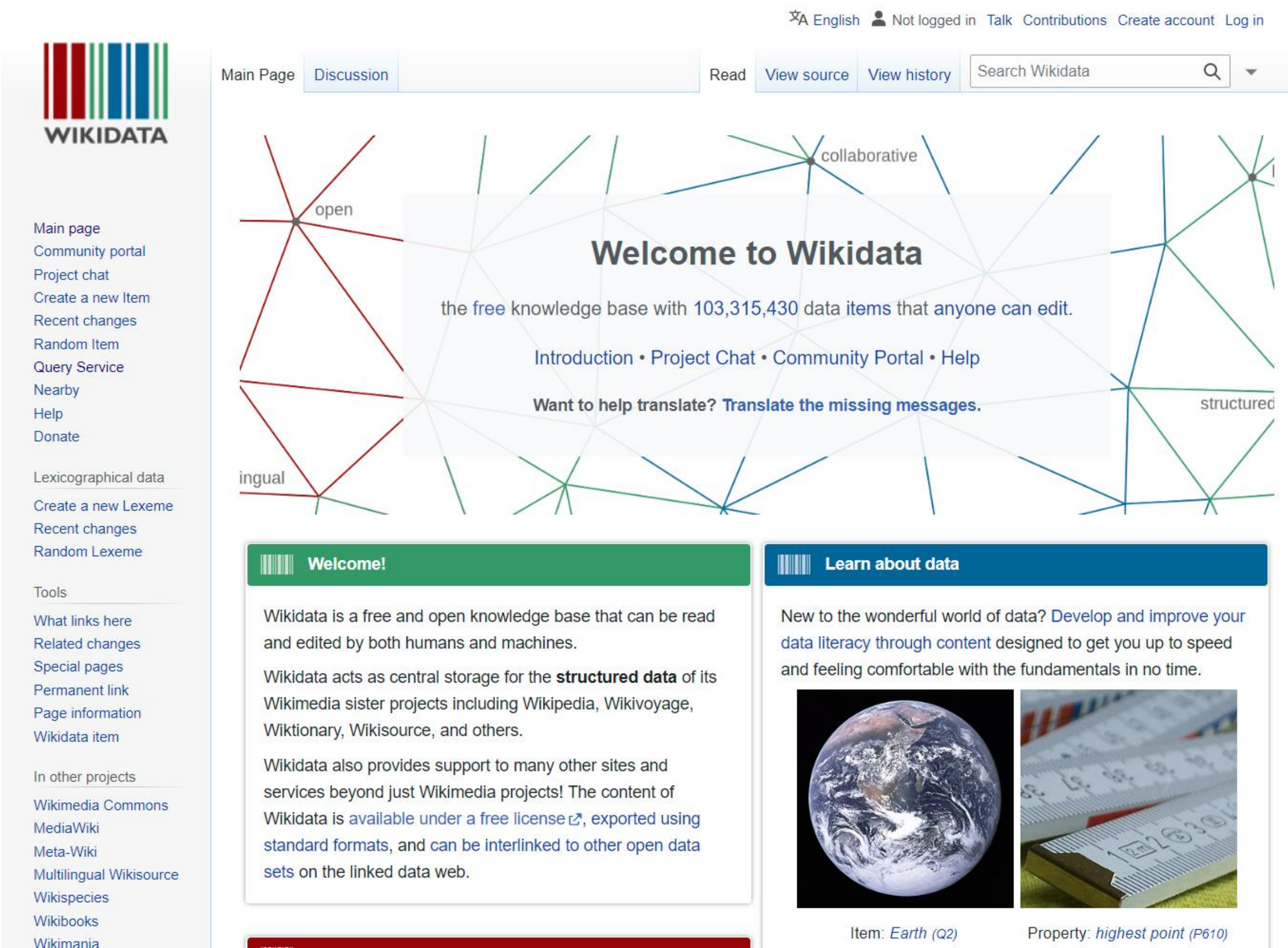
¿How to implement a Graph Database?

Part 1:
What are Graph Databases?


Google The Keyword
Latest stories
Product updates
Company news
Linked in Engineering
Home
Blog
Data
Open Source
Trust
Infrastructure
Microsoft | The AI Blog
The Official Microsoft Blog
Microsoft On the Issues
Transform
All Microsoft
ebay
Company
Stories
Impact
Tech
Investors
Careers
Press Room
Follow Us
Contact Us
Uber Engineering
Blog
Research
Tech Offices
Airbnb Engineering & Data Science
AI & ML
DATA
WEB
MOBILE
INFRASTRUCTURE
FINTECH
PEOPLE
amazon | science
Latest news
Research areas
Conferences & events
Publications
Academic engagements
Blog
Careers
INFORMATION AND KNOWLEDGE MANAGEMENT
Combining knowledge graphs, quickly and accurately
Novel cross-graph-attention and self-attention mechanisms enable state-of-the-art performance.
By Hao Wei
March 19, 2020
Knowledge graphs are a way of representing information that can capture complex relationships more easily than conventional databases. At Amazon, we use knowledge graphs to represent the hierarchical relationships between product types on amazon.com; the relationships between creators and content on Amazon Music and Prime Video; and general information for Alexa's question-answering service — among other things.
RELATED PUBLICATIONS
Collective Knowledge Graph Multi-type Entity Alignment
Qi Zhu, Hao Wei, Bunyamin Sisman, Da Zheng, Christos Faloutsos, Xin Luna Dong, Jiawei Han
2020
INFORMATION AND KNOWLEDGE MANAGEMENT
Download
CONFERENCE / JOURNAL
The Web Conference 2020
RECENT BLOG POSTS
How SageMaker's algorithms help democratize machine learning
Zohar Karnin
June 24, 2020

An example of a “knowledge graph”?

Wikidata: Wikipedia but with graph data



What kinds of entities?

WIKIDATA

Main page
Community portal
Project chat
Create a new Item
Recent changes
Random Item
Query Service
Nearby
Help
Donate

Lexicographical data

Create a new Lexeme
Recent changes
Random Lexeme

Tools

What links here
Related changes
Special pages
Permanent link
Page information
Concept URI
Cite this page
Get shortened URL
Download QR code

Item | Discussion

Geoffrey Hinton (Q92894)

British-Canadian computer scientist and psychologist

edit

Geoffrey Everest Hinton | Geoff Hinton | Geoffrey E. Hinton | G. E. Hinton

In more languages

Configure

Language	Label	Description	Also known as
English	Geoffrey Hinton	British-Canadian computer scientist and psychologist	Geoffrey Everest Hinton Geoff Hinton Geoffrey E. Hinton G. E. Hinton
Spanish	Geoffrey Hinton	informático y psicólogo británico-canadiense	Geoffrey Everest Hinton Geoffrey E. Hinton
Mapuche	Geoffrey Hinton	No description defined	
default for all languages	Geoffrey Hinton	–	Geoffrey Everest Hinton Geoffrey E. Hinton

All entered languages

Statements

instance of	human 1 reference	edit
		add value

What kinds of entities?

WIKIDATA

Main page
Community portal
Project chat
Create a new Item
Recent changes
Random Item
Query Service
Nearby
Help
Donate

Lexicographical data

Create a new Lexeme
Recent changes
Random Lexeme

Tools

What links here
Related changes
Special pages
Permanent link
Page information
Concept URI
Cite this page
Get shortened URL
Download QR code

Item | Discussion

Maryland (Q1391)

state of the United States of America

State of Maryland | Maryland, United States | MD | Md. | Old Line State | US-MD

Statements

instance of	U.S. state 4 references
part of	contiguous United States 1 reference South Atlantic states 2 references Mid-Atlantic 1 reference
inception	28 April 1788 *Gregorian* 5 references

What kinds of entities?

WIKIDATA

Item | Discussion

Main page
Community portal
Project chat
Create a new Item
Recent changes
Random Item
Query Service
Nearby
Help
Donate

Lexicographical data

Create a new Lexeme
Recent changes
Random Lexeme

Tools

What links here
Related changes
Special pages
Permanent link
Page information
Concept URI
Cite this page
Get shortened URL
Download QR code

crab cake (Q1138371)

Statements

instance of	dish 0 references	edit + add reference + add value
subclass of	crab dish 0 references	edit + add reference + add value
image		edit

What kinds of entities?

WIKIDATA

Main page
Community portal
Project chat
Create a new Item
Recent changes
Random Item
Query Service
Nearby
Help
Donate

Lexicographical data

Create a new Lexeme
Recent changes
Random Lexeme

Tools

What links here
Related changes
Special pages
Permanent link
Page information
Concept URI
Cite this page
Get shortened URL
Download QR code

Item | Discussion

Sharknado (Q13794921)

2013 film directed by Anthony C. Ferrante

edit

In more languages

Statements

instance of	television film 1 reference	edit + add value
logo image	SHARKNADO Sharknado logo.png 1,281 × 471; 693 KB 0 references	edit + add reference + add value
title	Sharknado (English)	edit

What kinds of entities?

WIKIDATA

Main page
Community portal
Project chat
Create a new Item
Recent changes
Random Item
Query Service
Nearby
Help
Donate

Lexicographical data

Create a new Lexeme
Recent changes
Random Lexeme

Tools

What links here
Related changes
Special pages
Permanent link
Page information
Concept URI
Cite this page
Get shortened URL
Download QR code

Item | Discussion

TRAPPIST-1 (Q23986556)

ultra-cool dwarf star

edit

2MASS J23062928-0502285 | Trappist 1

In more languages

Statements

instance of	red dwarf	edit
	0 references	
		add reference
	ultra-cool dwarf	edit
	0 references	
		add reference
	infrared source	edit
	1 reference	
	high proper-motion star	edit
	1 reference	
	low-mass star	edit

Where is Wikidata used?

TRAPPIST

Article Talk

From Wikipedia, the free encyclopedia

(Redirected from Transiting Planets and Planetesimals Small Telescope)

Not to be confused with Trappists.

The **Transiting Planets and Planetesimals Small Telescope** (**TRAPPIST**) is the corporate name for a pair of Belgian optic robotic telescopes. **TRAPPIST–South**, which is situated high in the Chilean mountains at ESO's La Silla Observatory, came online in 2010, and **TRAPPIST–North** situated at the Oukaïmeden Observatory in the Atlas Mountains in Morocco, came online in 2016.[1]

Description [edit]

TRAPPIST is controlled from Liège, Belgium, with some autonomous features. It consists of two 60 cm (24 in) reflecting robotic telescopes located at the ESO La Silla Observatory (housed in the dome of the retired **Swiss T70 telescope**) in Chile and at Oukaïmeden Observatory in Morocco.

The 60 cm f/8 Ritchey–Chrétien design telescopes and New Technology Mount NTM-500 were built by ASTELCO Systems, a company in Germany. The CCD camera was built by Finger Lakes Instrumentation (USA), providing a 22 x 22 arcminutes field of view. The camera is fitted with a double filter wheel, allowing 12 different filters and one clear position.[2][3]

The telescope condominium is a joint venture between the University of Liège, Belgium, and Geneva Observatory, Switzerland, and among other tasks, it specializes in searching for comets and exoplanets.[4][5]

TRAPPIST

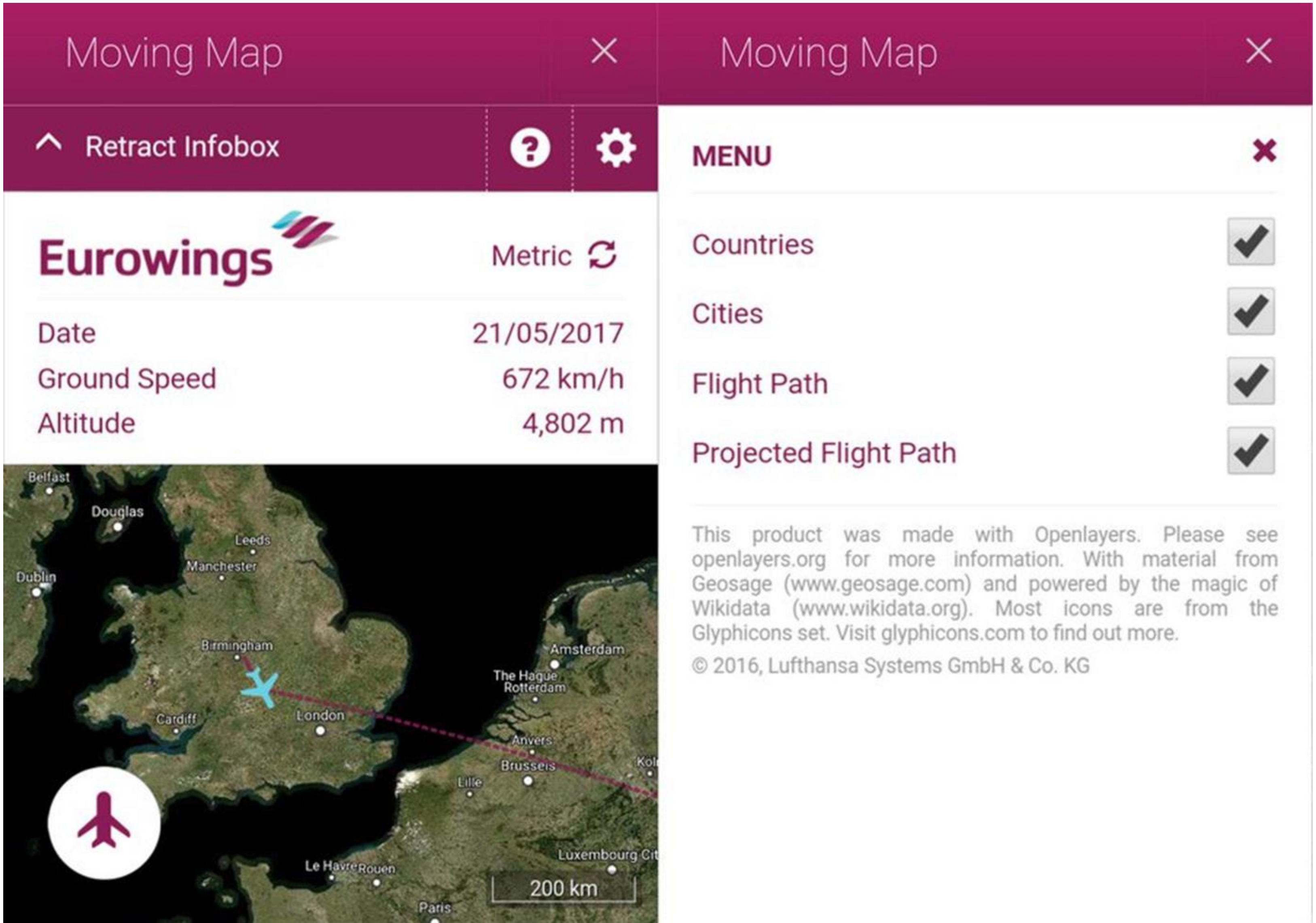
Part of	La Silla Observatory Oukaïmeden Observatory
Location(s)	Coquimbo Region, Chile
Coordinates	29°15′17″S 70°44′22″W
Organization	University of Liège
Observatory code	I40
Altitude	2,400 m (7,900 ft)
Telescope style	Robotic optical telescope
Website	www.trappist.uliege.be

Location of TRAPPIST

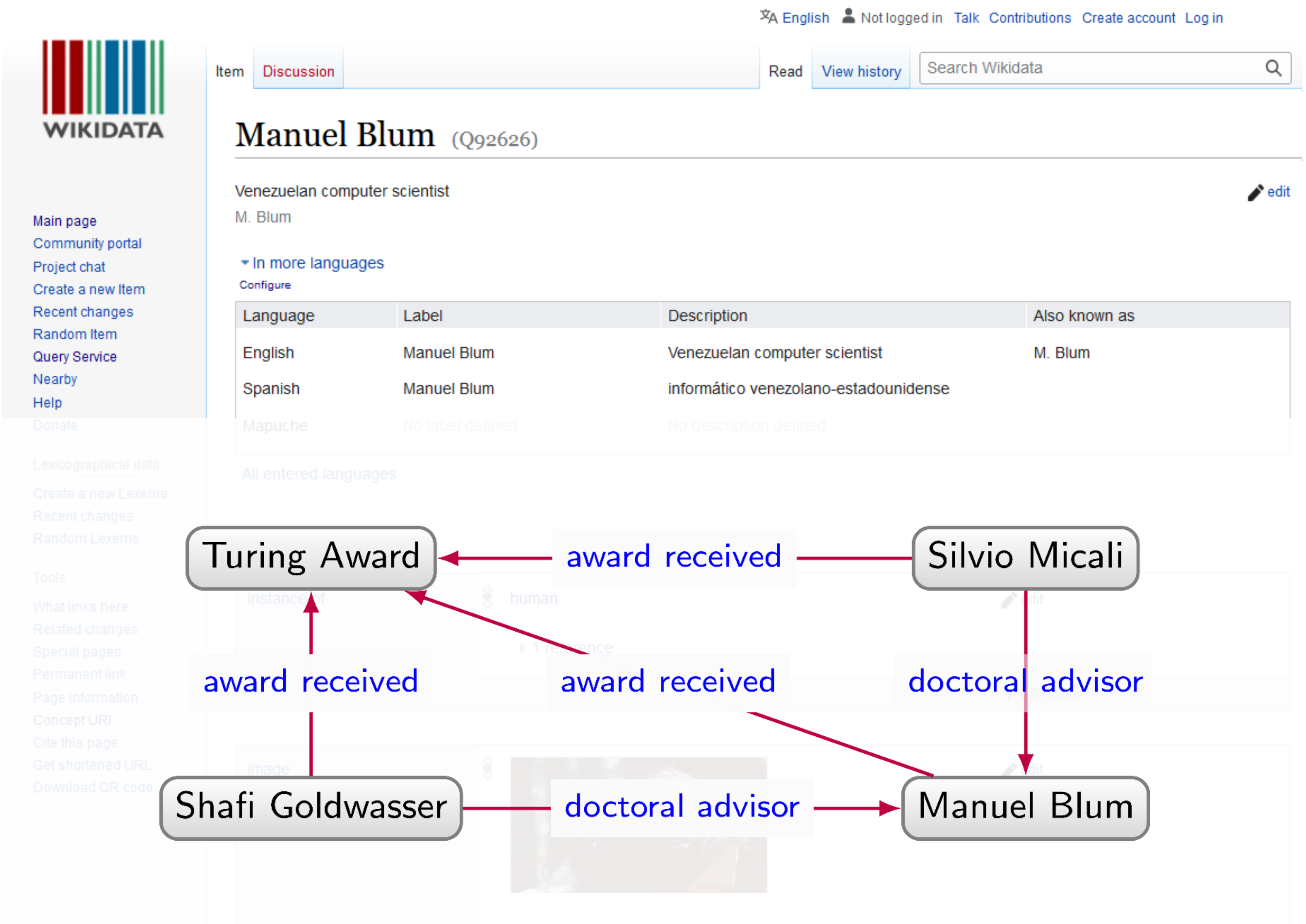
Related media on Commons

[edit on Wikidata]

Where is Wikidata used?



How is this a graph?



Knowledge Graph Management: Graph Databases

Popular graph databases

neo4j

OrientDB®

Dgraph

JanusGraph

ArangoDB

TigerGraph

Amazon Neptune

MK/DB>

blazegraph™

Jena

AllegroGraph

OPENLINK® VIRTUOSO UNIVERSAL SERVER

STARDOG

GraphDB

Popular graph databases

https://db-engines.com/

DB-Engines Ranking

423 systems in ranking, October 2024

Rank Oct 2024	Rank Sep 2024	Rank Oct 2023	DBMS	Database Model	Score Oct 2024	Score Sep 2024	Score Oct 2023
1.	1.	1.	Oracle	Relational, Multi-model	1309.45	+22.85	+48.03
2.	2.	2.	MySQL	Relational, Multi-model	1022.76	-6.73	-110.56
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	802.09	-5.67	-94.79
4.	4.	4.	PostgreSQL	Relational, Multi-model	652.16	+7.80	+13.34
5.	5.	5.	MongoDB	Document, Multi-model	405.21	-5.02	-26.21
6.	6.	6.	Redis	Key-value, Multi-model	149.63	+0.20	-13.33
7.	7.	↑ 11.	Snowflake	Relational	140.60	+6.88	+17.36
8.	8.	↓ 7.	Elasticsearch	Multi-model	131.85	+3.06	-5.30
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model	122.77	-0.28	-12.10
10.	10.	↓ 9.	SQLite	Relational	101.91	-1.43	-23.23
11.	11.	↑ 12.	Apache Cassandra	Wide column, Multi-model	97.61	-1.34	-11.21
12.	12.	↓ 10.	Microsoft Access	Relational	92.15	-1.61	-32.16
13.	13.	↑ 14.	Splunk	Search engine	91.27	-1.75	-1.10
14.	14.	↑ 17.	Databricks	Multi-model	85.60	+1.35	+9.78
15.	15.	↓ 13.	MariaDB	Relational, Multi-model	84.89	+1.45	-14.77
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-model	74.53	+1.58	-6.40
17.	17.	↓ 16.	Amazon DynamoDB	Multi-model	71.85	+1.78	-9.07
18.	18.	18.	Apache Hive	Relational	52.57	-0.50	-16.61
19.	19.	↑ 20.	Google BigQuery	Relational	51.18	-1.48	-5.39
20.	20.	↑ 21.	FileMaker	Relational	44.40	-0.80	-8.92
21.	21.	↑ 23.	Neo4j	Graph	42.51	-0.17	-5.93

Popular graph databases

https://db-engines.com/

DB-Engines Ranking of Graph DBMS

☐ include secondary database models

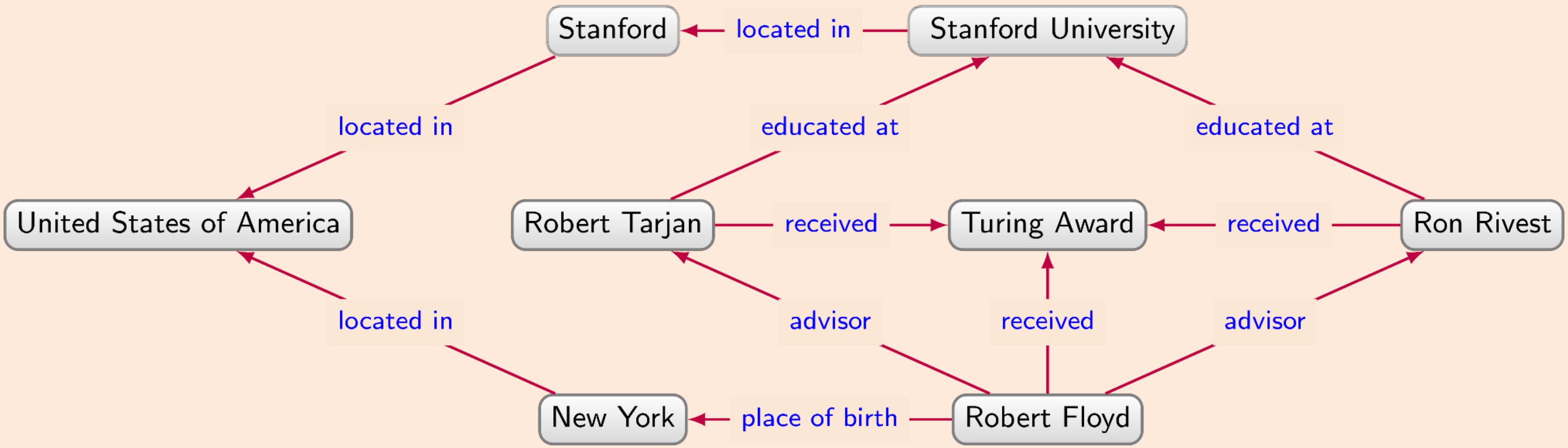
43 systems in ranking, October 2024

Rank Oct 2024	Rank Sep 2024	Rank Oct 2023	DBMS	Database Model	Score Oct 2024	Score Sep 2024	Score Oct 2023
1.	1.	1.	Neo4j	Graph	42.51	-0.17	-5.93
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	24.50	-0.47	-9.80
3.	3.	3.	Aerospike	Multi-model	5.57	+0.41	-0.86
4.	4.	4.	Virtuoso	Multi-model	3.91	-0.08	-1.51
5.	5.	↑ 6.	ArangoDB	Multi-model	3.44	+0.13	-0.83
6.	6.	↓ 5.	OrientDB	Multi-model	3.03	+0.01	-1.24
7.	7.	7.	Memgraph	Graph	2.82	-0.09	+0.01
8.	8.	↑ 9.	GraphDB	Multi-model	2.77	+0.01	+0.19
9.	9.	↑ 10.	Amazon Neptune	Multi-model	2.17	-0.03	-0.37
10.	10.	↑ 12.	Stardog	Multi-model	1.92	-0.01	-0.34
11.	11.	↓ 8.	NebulaGraph	Graph	1.86	-0.06	-0.91
12.	12.	↓ 11.	JanusGraph	Graph	1.78	-0.07	-0.52
13.	13.	↑ 14.	Fauna	Multi-model	1.50	-0.05	-0.39
14.	14.	↓ 13.	TigerGraph	Graph	1.46	+0.02	-0.64
15.	15.	15.	Dgraph	Graph	1.39	0.00	-0.47
16.	16.	16.	Giraph	Graph	1.11	-0.02	-0.60
17.	17.	↑ 19.	SurrealDB	Multi-model	1.07	-0.04	+0.01
18.	18.	↓ 17.	AllegroGraph	Multi-model	0.80	-0.04	-0.40
19.	19.	↓ 18.	Blazegraph	Multi-model	0.74	-0.01	-0.34
20.	20.	20.	TypeDB	Multi-model	0.66	+0.01	-0.38

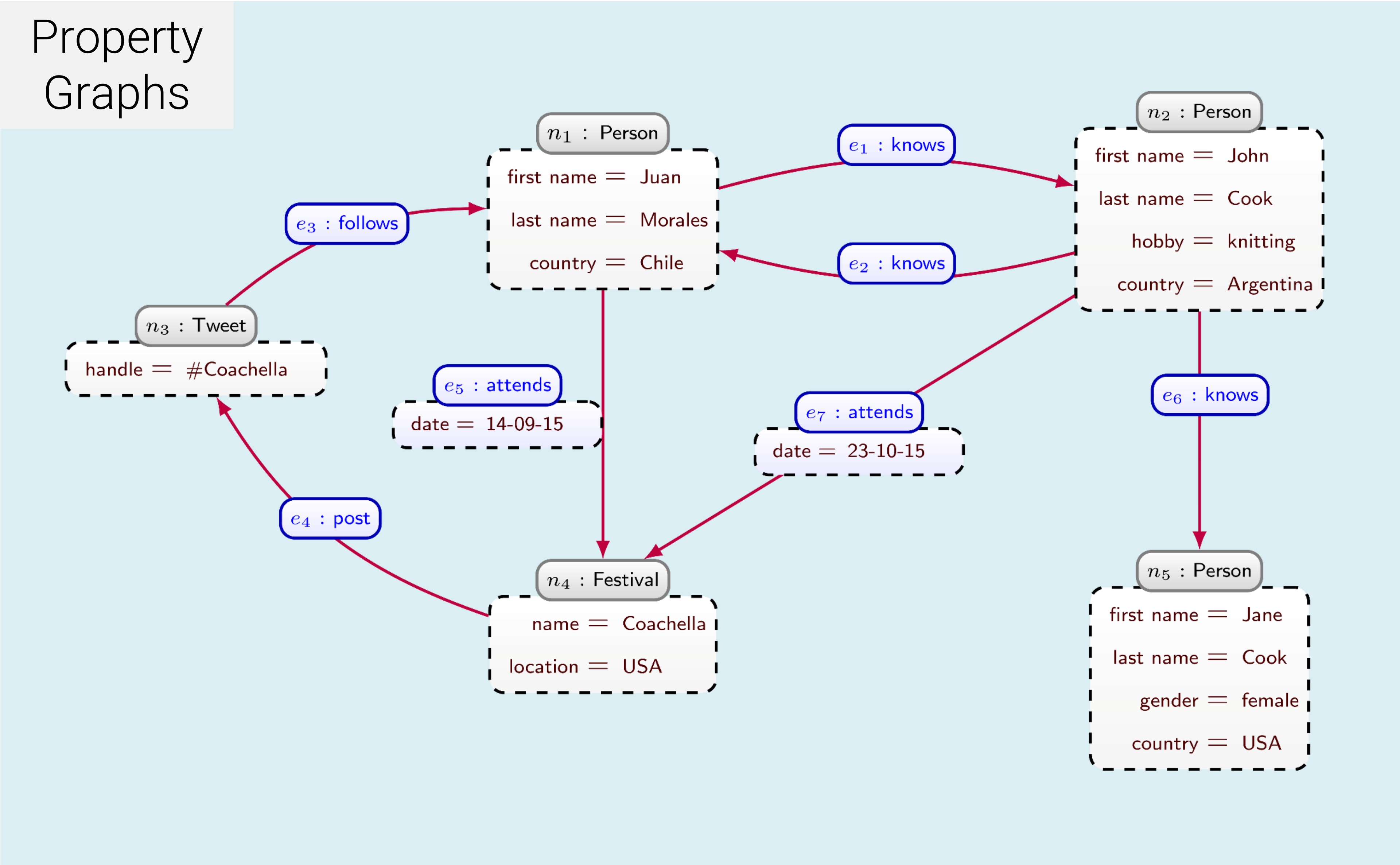
Graph Databases:
Data Models

Directed edge-labelled graph (RDF)

RDF

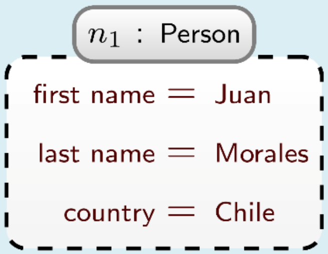


Property graphs

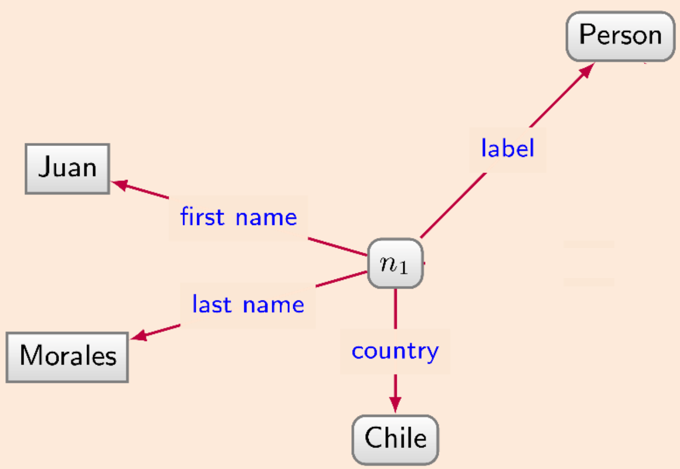


Property graphs vs RDF

Property Graphs

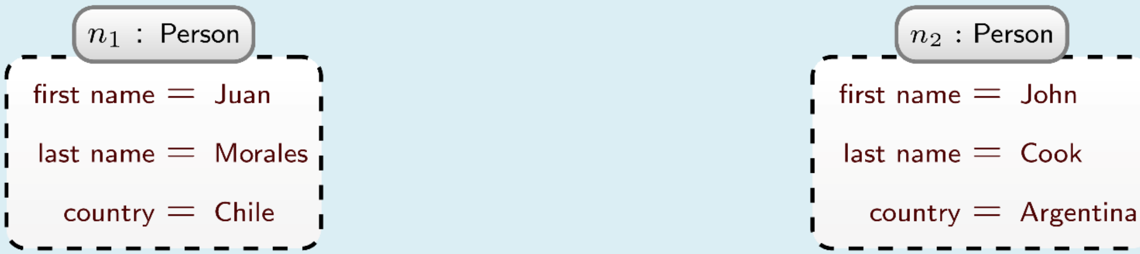


RDF

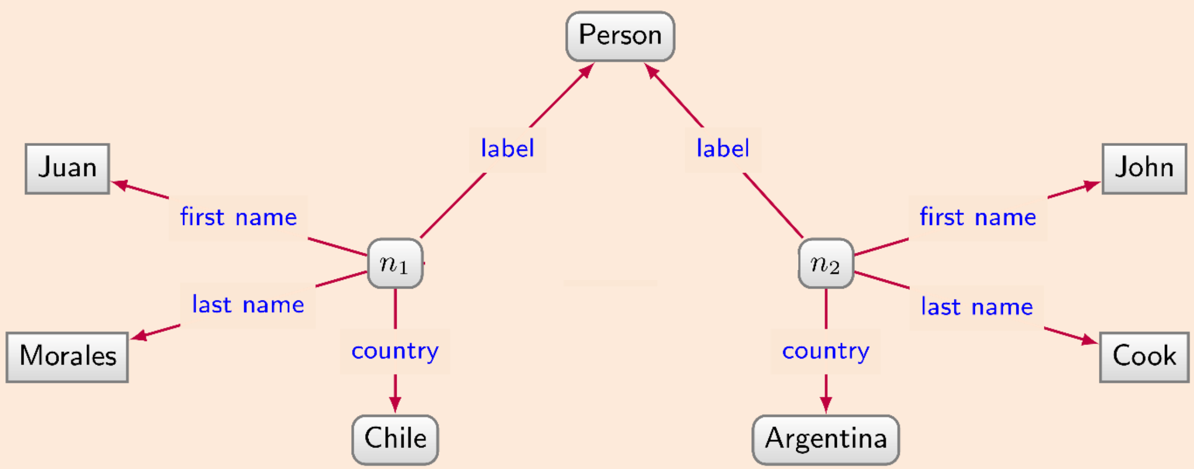


Property graphs vs RDF

Property Graphs

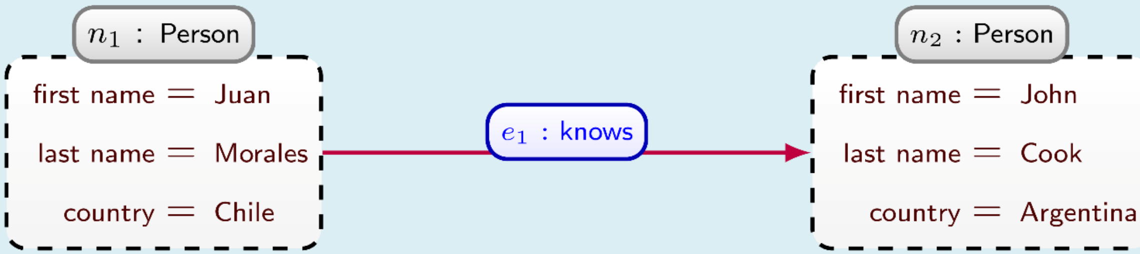


RDF

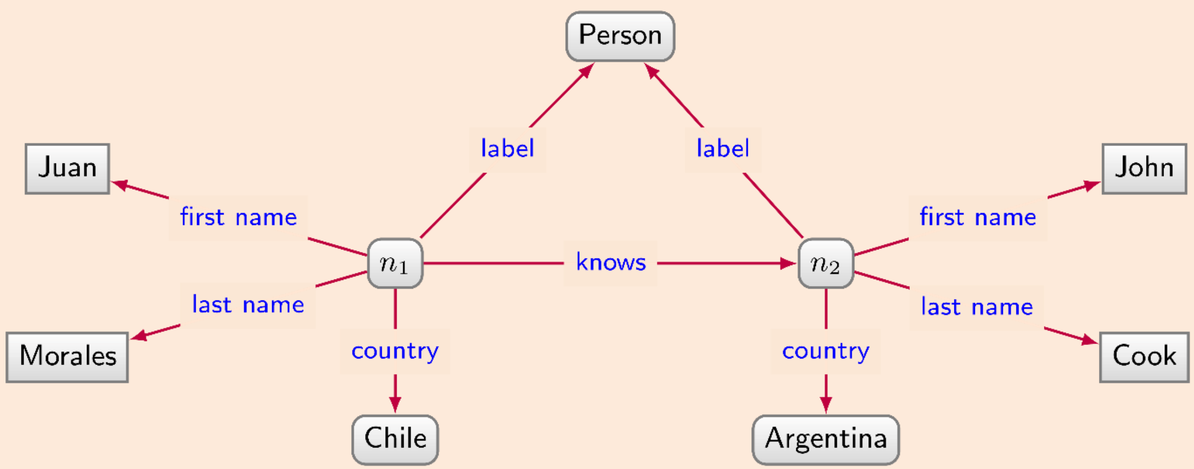


Property graphs vs RDF

Property Graphs

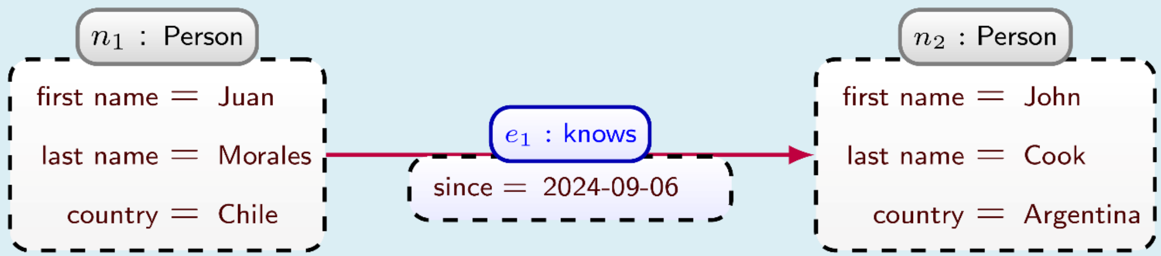


RDF

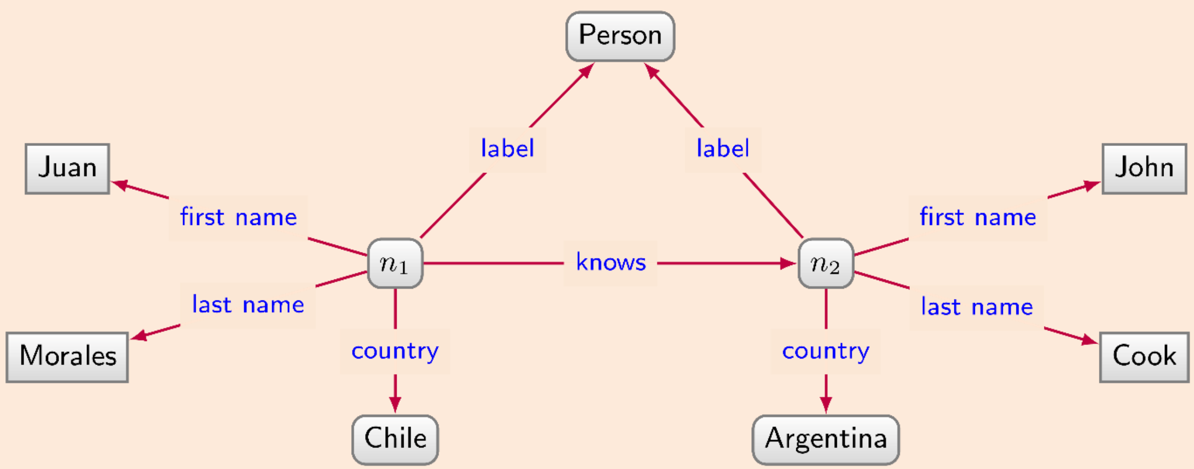


Property graphs vs RDF

Property Graphs

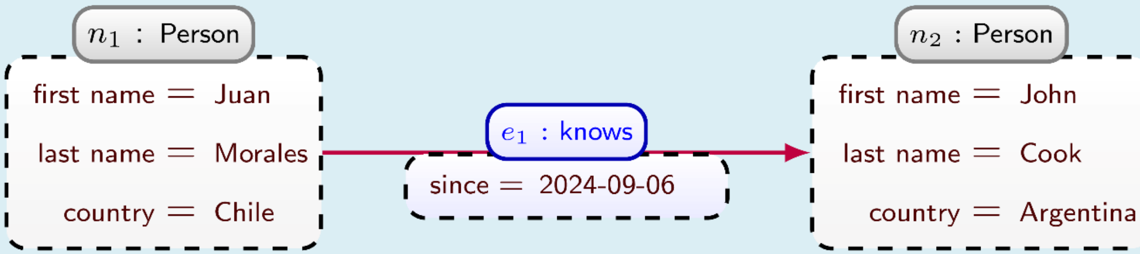


RDF

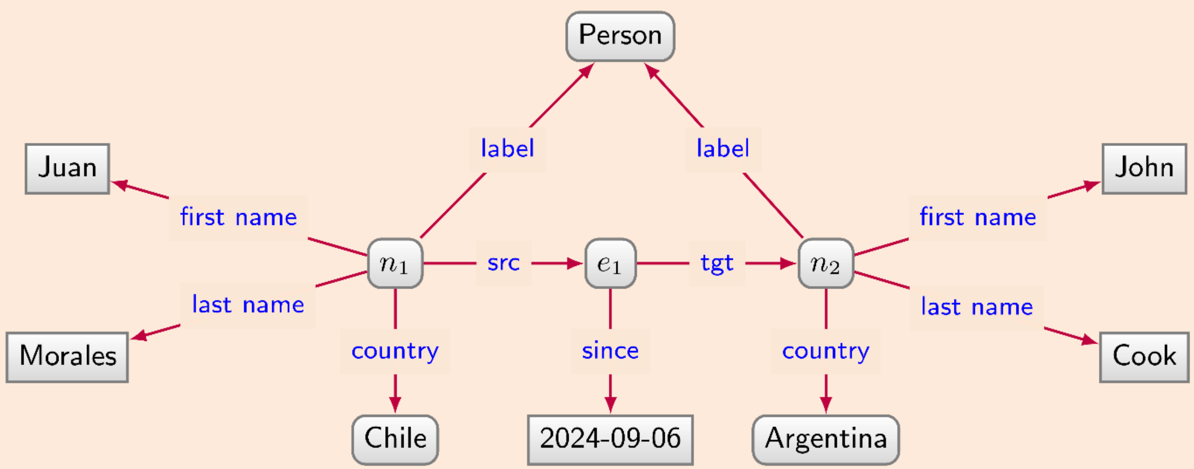


Property graphs vs RDF

Property Graphs



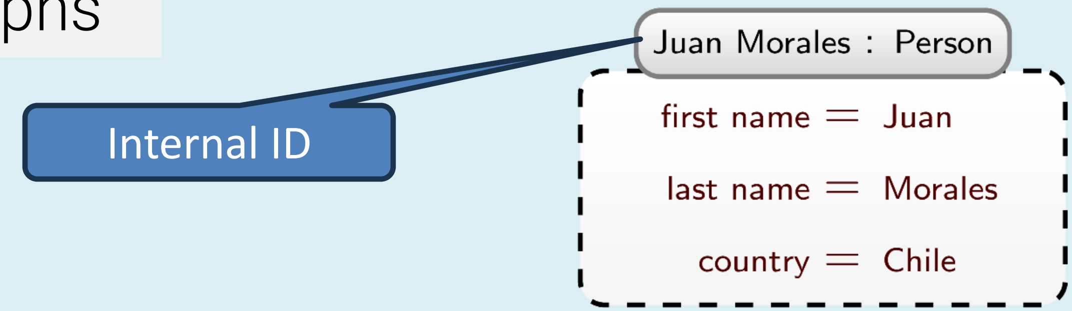
RDF



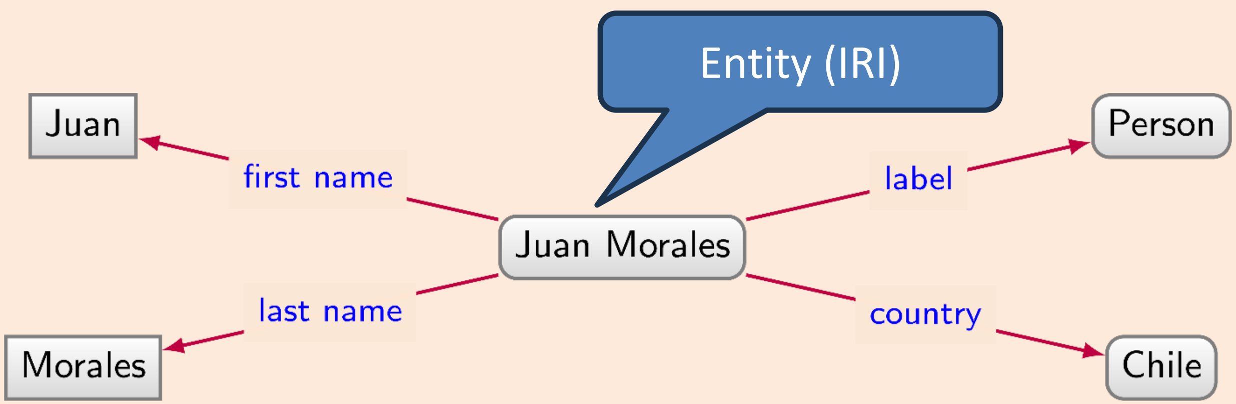
See [Reification] for details

Property graphs vs RDF: the “node”

Property Graphs



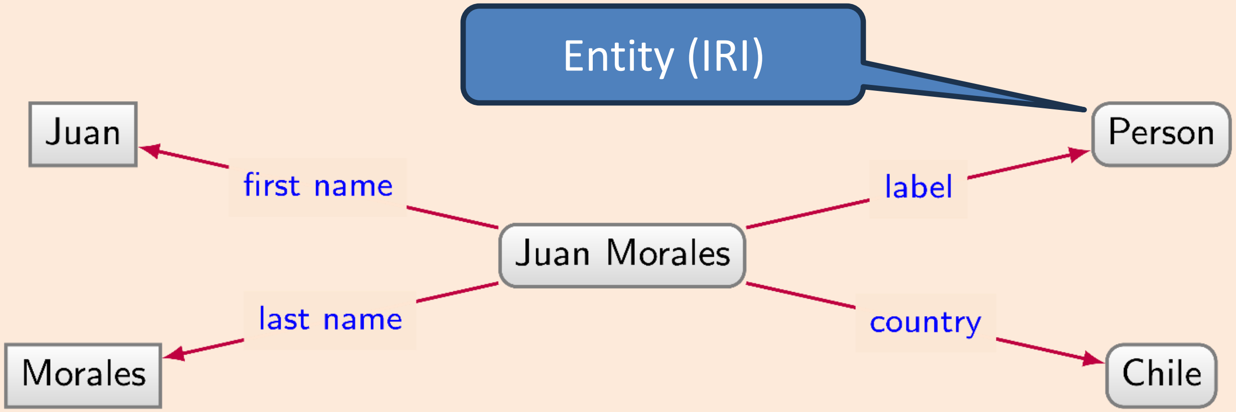
RDF



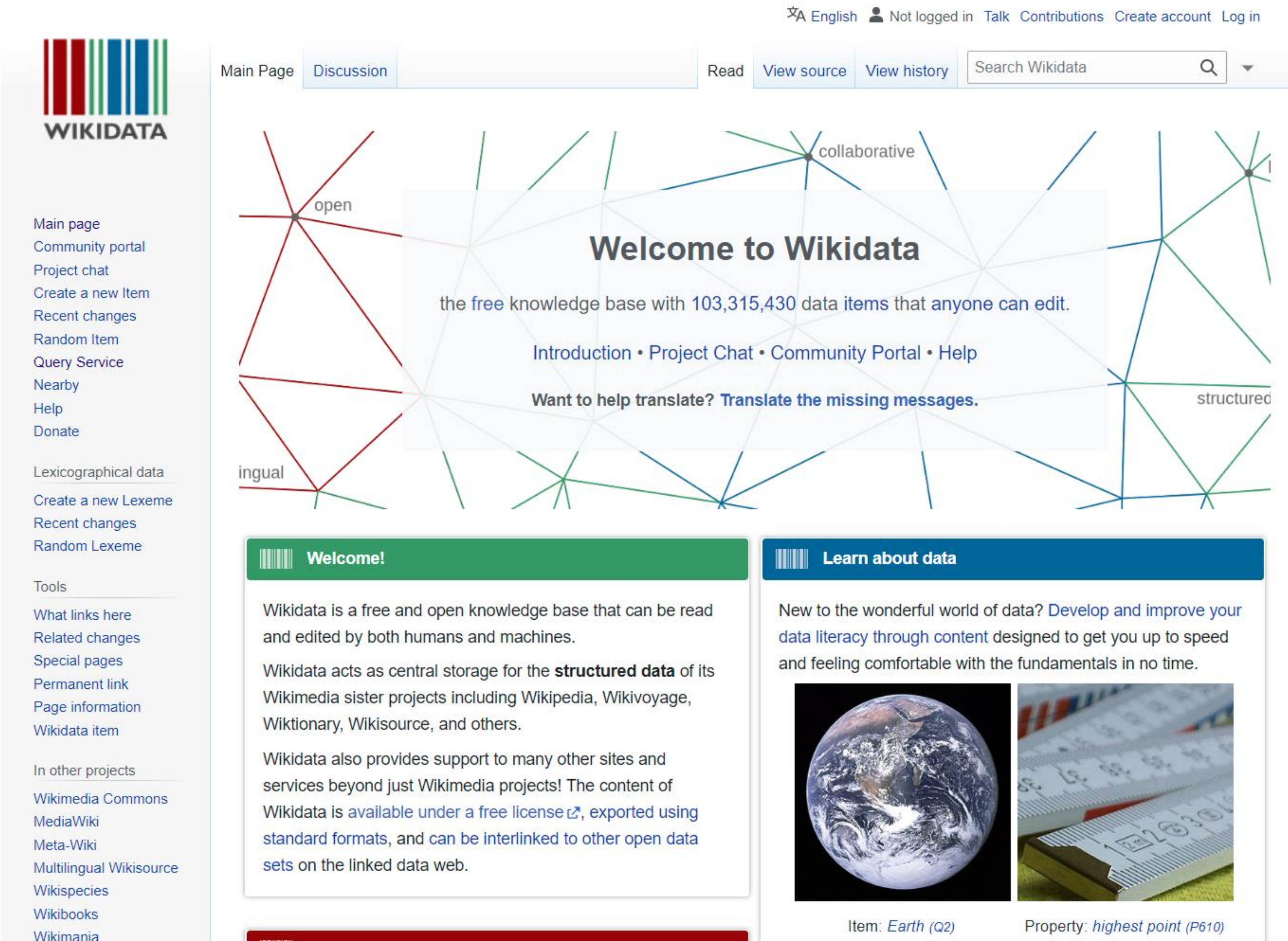
Property graphs vs RDF: the "node"

Property Graphs

RDF



Wikidata: Wikipedia but with graph data



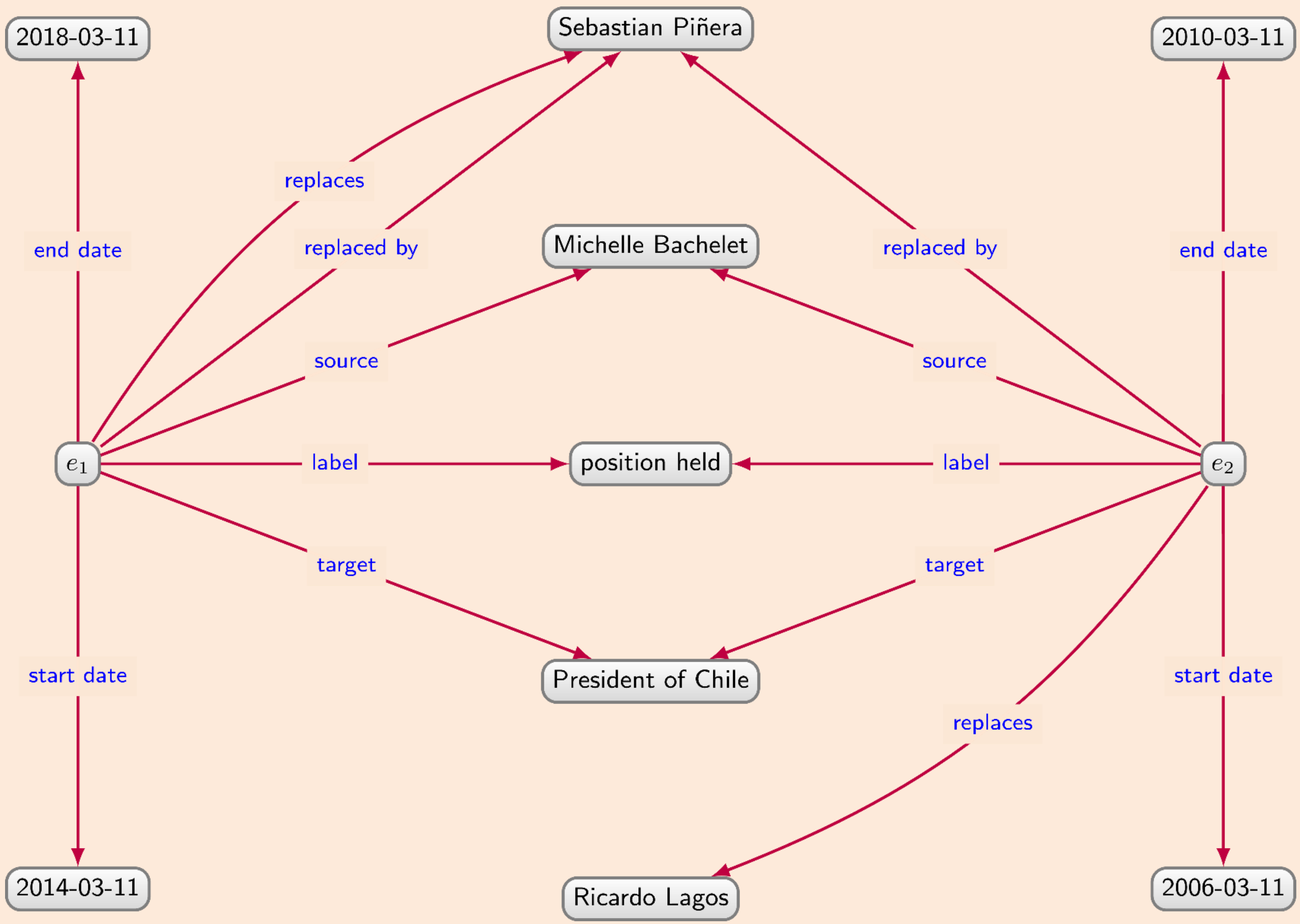
Wikidata statements

Michelle Bachelet [Q320]

position held [P39]		President of Chile [Q466956]
	start date [P580]	2014-03-11
	end date [P582]	2018-03-11
	replaces [P155]	Sebastián Piñera [Q306]
	replaced by [P156]	Sebastián Piñera [Q306]
position held [P39]		President of Chile [Q466956]
	start date [P580]	2006-03-11
	end date [P582]	2010-03-11
	replaces [P155]	Ricardo Lagos [Q331]
	replaced by [P156]	Sebastián Piñera [Q306]

Can you represent this in RDF?

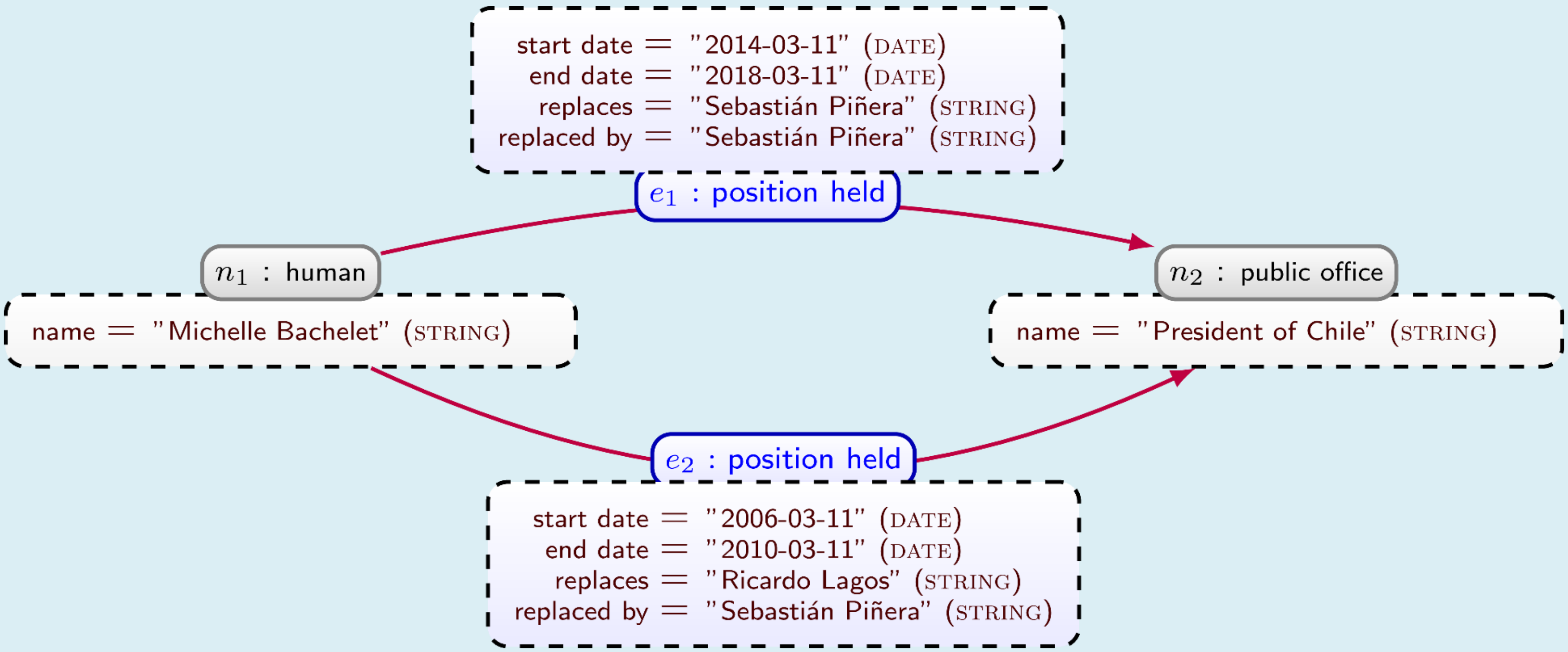
RDF



See [Reification] for details

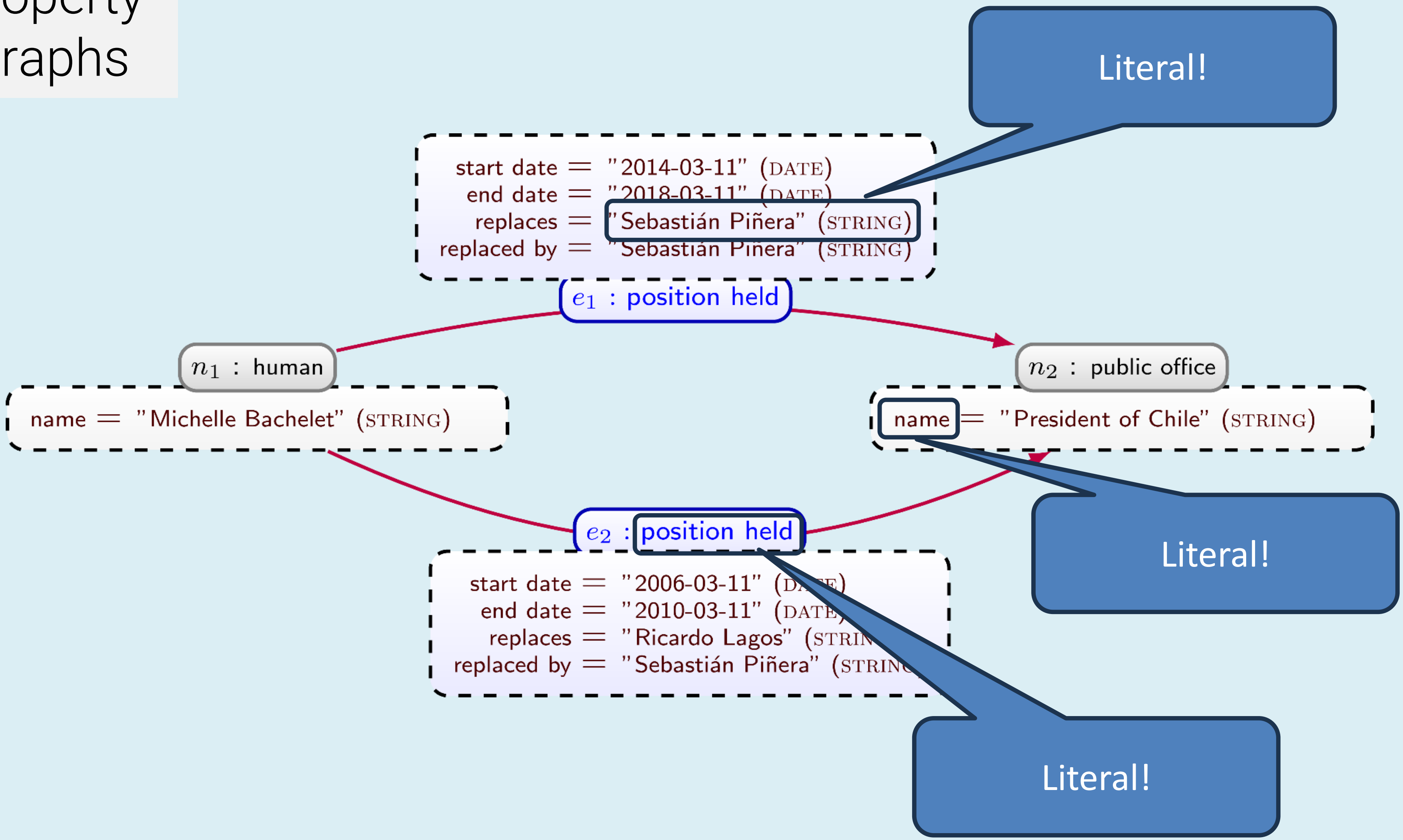
Property graphs

Property Graphs



Are Property graphs enough?

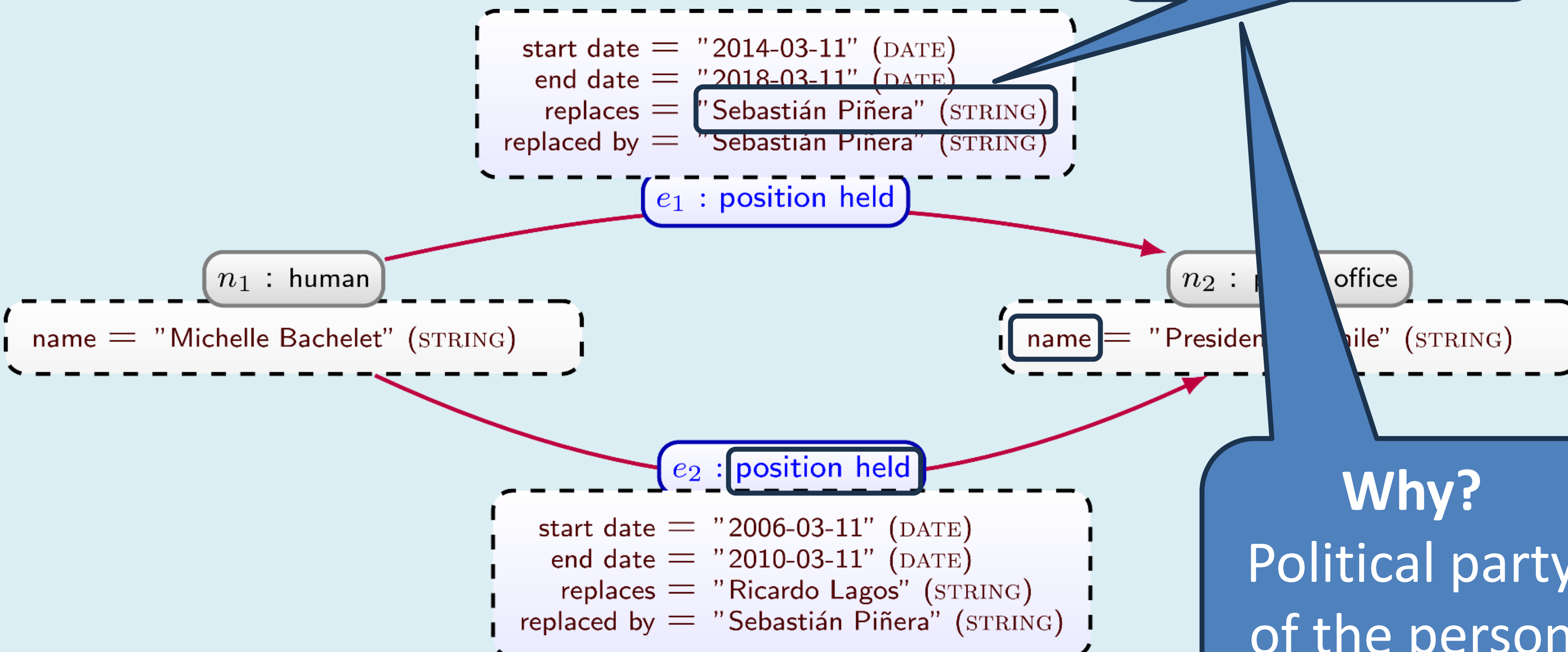
Property Graphs



Are Property graphs enough?

Property Graphs

Clickable!



Why?
Political party of the person replacing her!

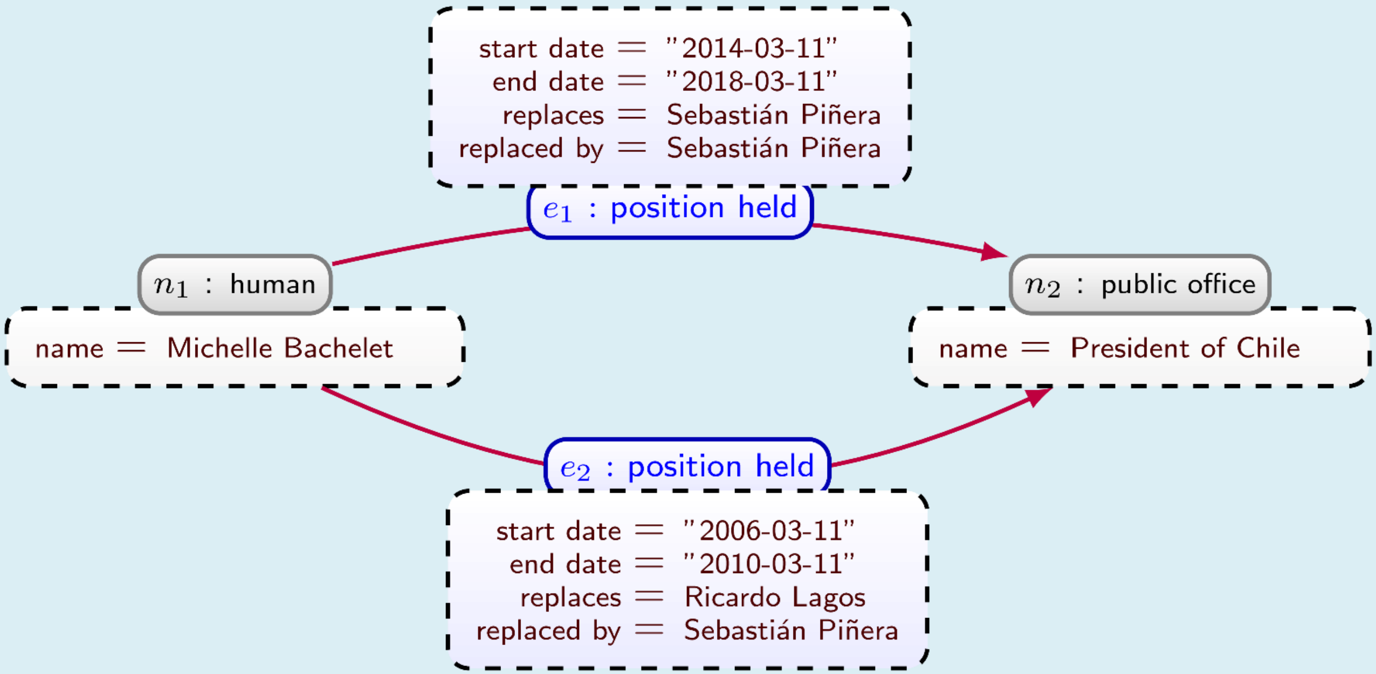
Solution: domain graphs

Domain graphs in a nutshell: make everything clickable

See [Multi22] for details

Solution: domain graphs

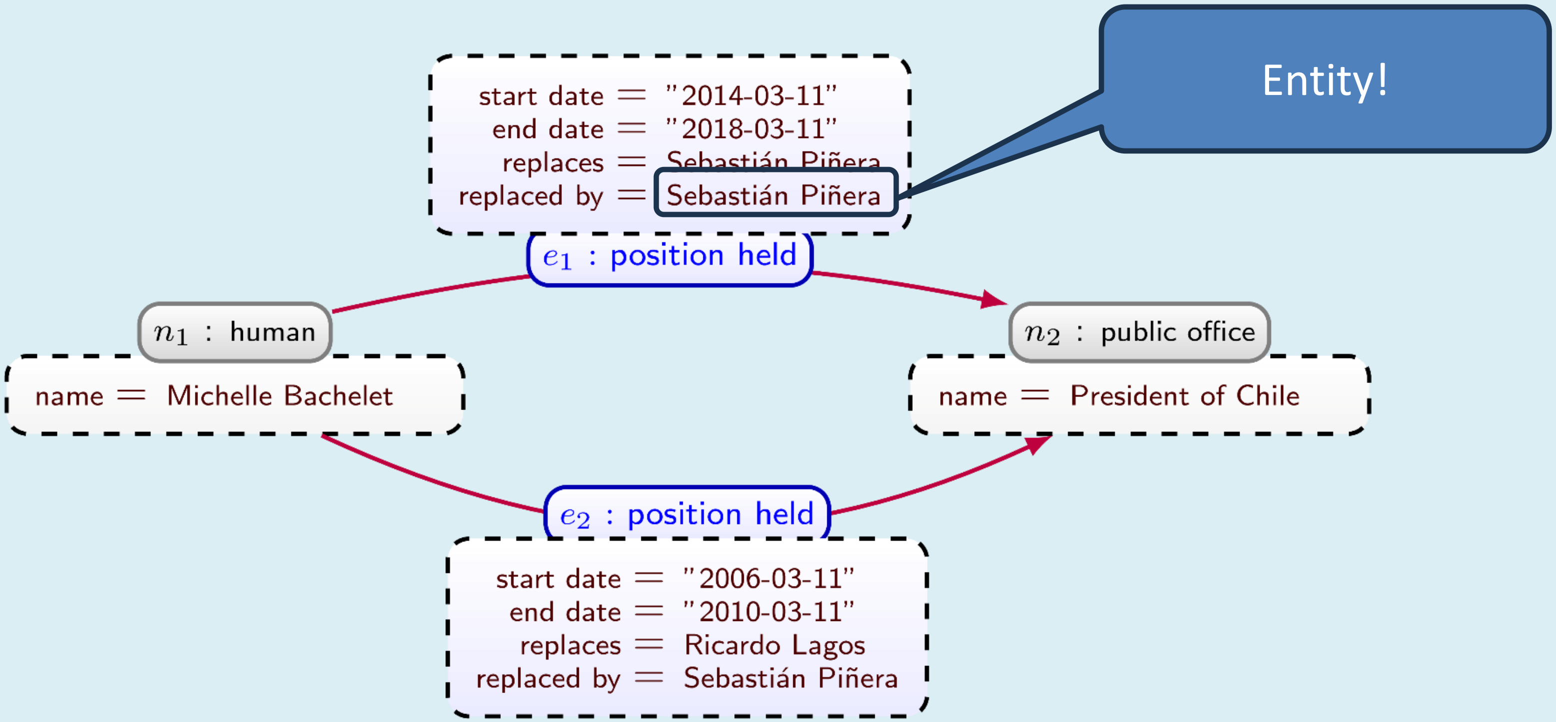
Domain graphs in a nutshell: make everything clickable



See [Multi22] for details

Solution: domain graphs

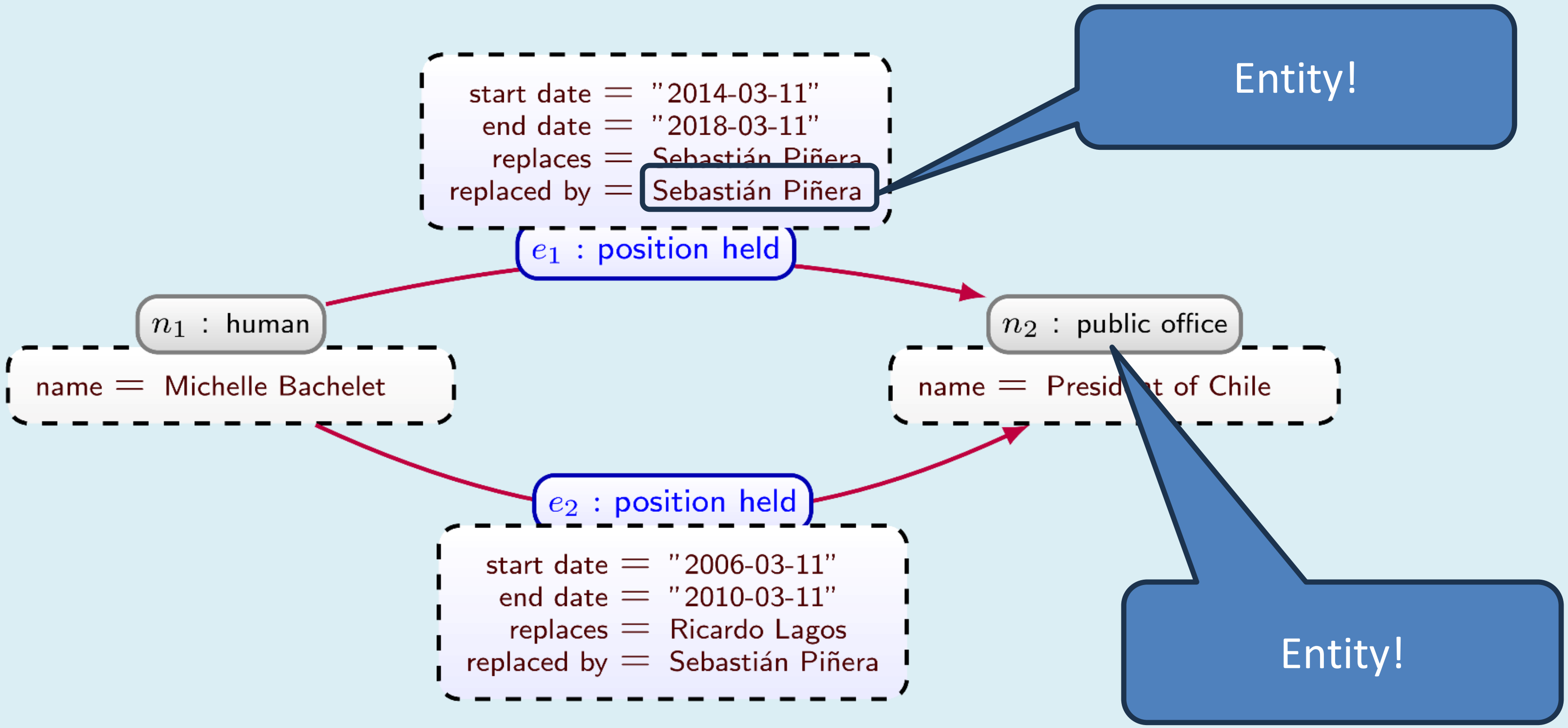
Domain graphs in a nutshell: make everything clickable



See [Multi22] for details

Solution: domain graphs

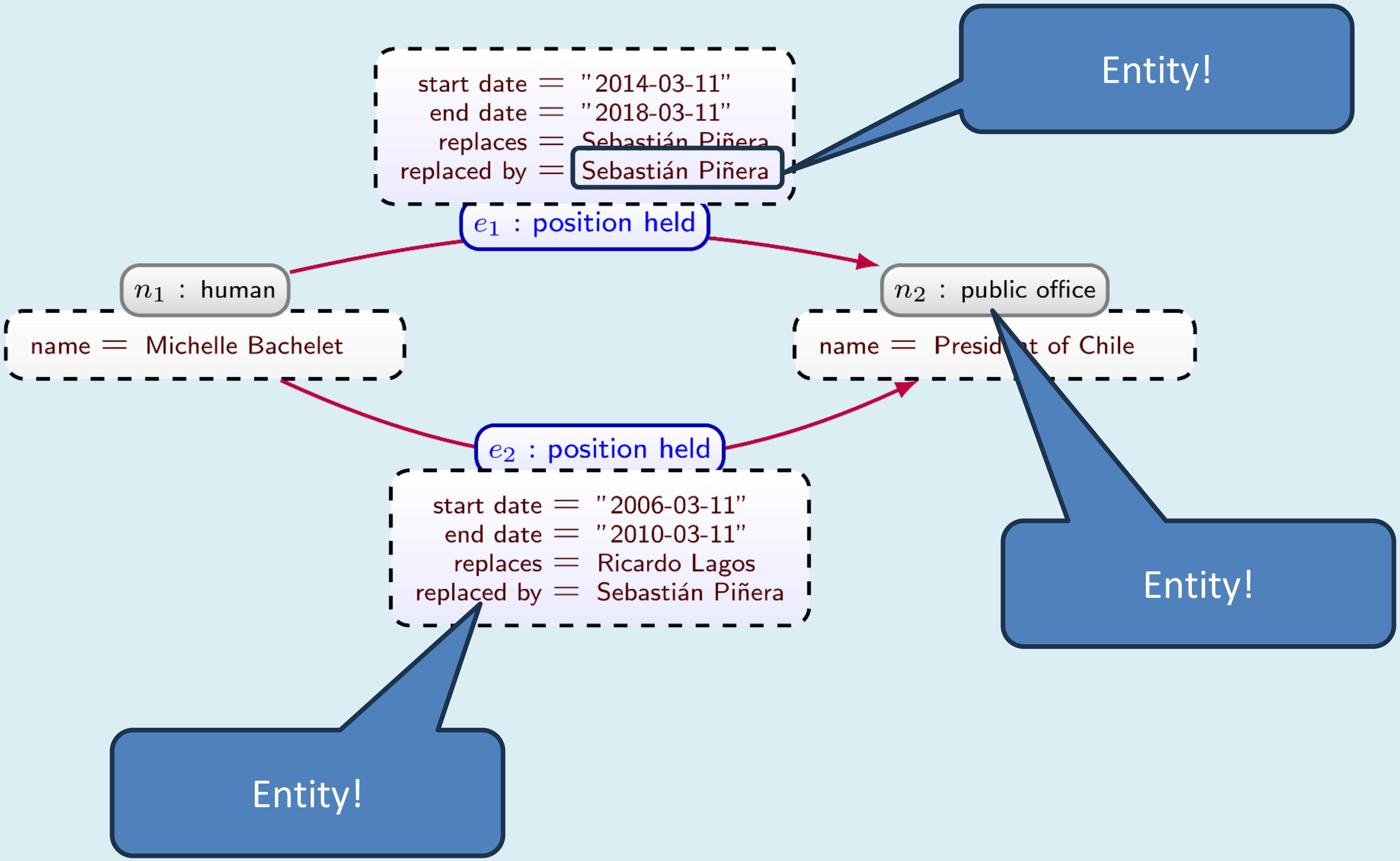
Domain graphs in a nutshell: make everything clickable



See [Multi22] for details

Solution: domain graphs

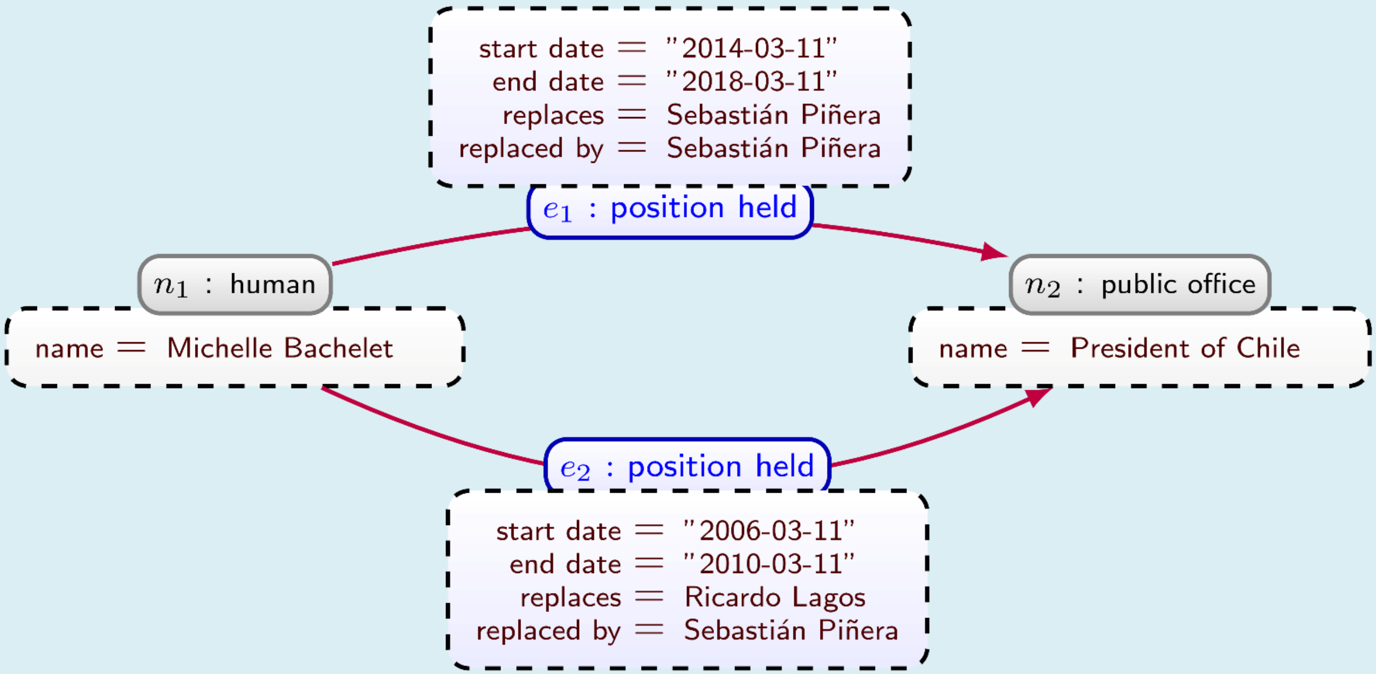
Domain graphs in a nutshell: make everything clickable



See [Multi22] for details

Solution: domain graphs

Domain graphs in a nutshell: make everything clickable

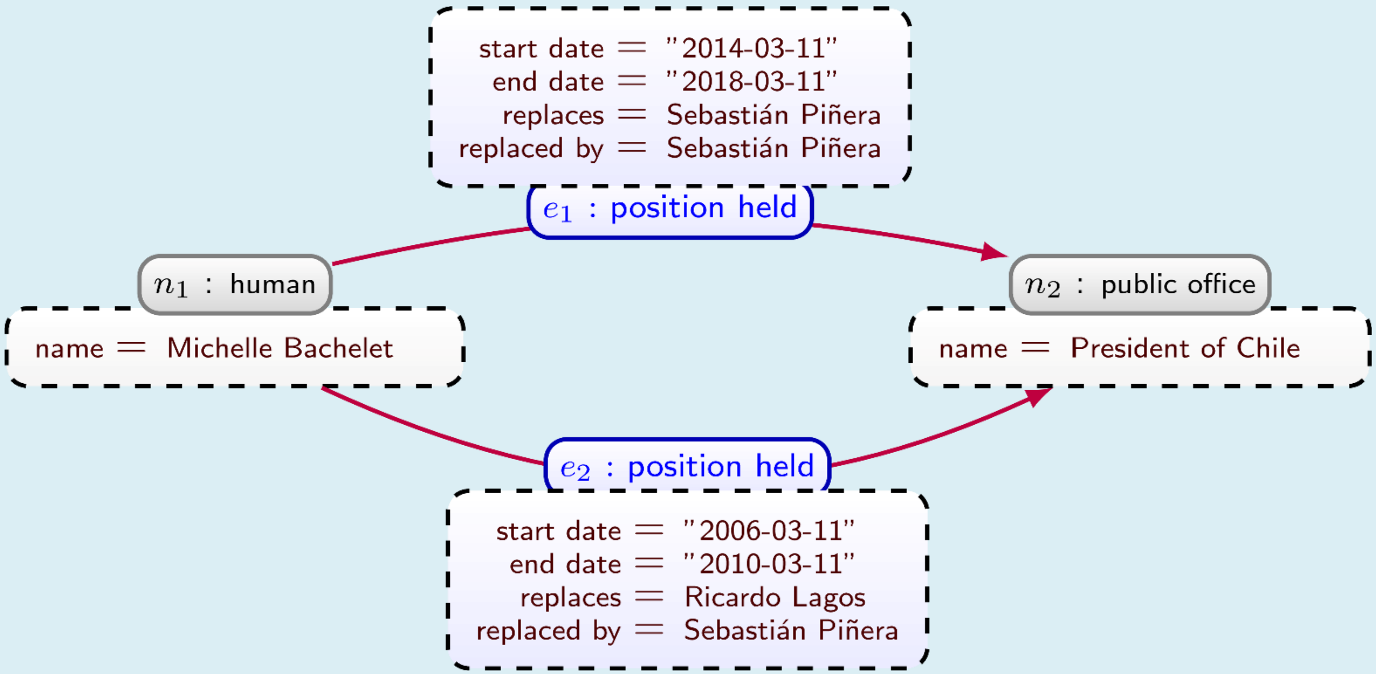


$$\textsc{DomainGraph}(\text{source}, \text{target}, \underline{\text{eid}})$$
$$\textsc{Labels}(\text{object}, \text{label})$$
$$\textsc{Properties}(\underline{\text{object}, \text{property}}, \text{value})$$

See [Multi22] for details

Implementing Domain Graphs

Perhaps this is enough: one label per edge?

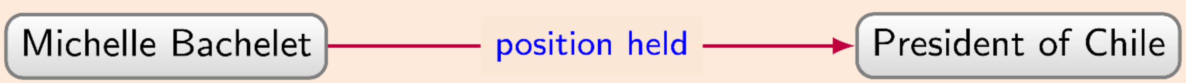


$$\textsc{DomainGraph}(\text{source}, \text{type}, \text{target}, \underline{\text{eid}})$$
$$\textsc{Labels}(\text{object}, \text{label})$$
$$\textsc{Properties}(\underline{\text{object}, \text{property}}, \text{value})$$

See [OneGraph,MDB] for details

Honourable mention: RDF*

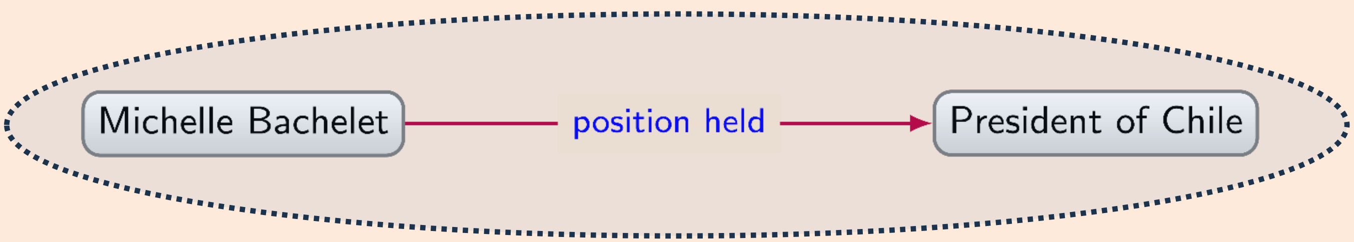
Quotable triples



See [RDF*] for details

Honourable mention: RDF*

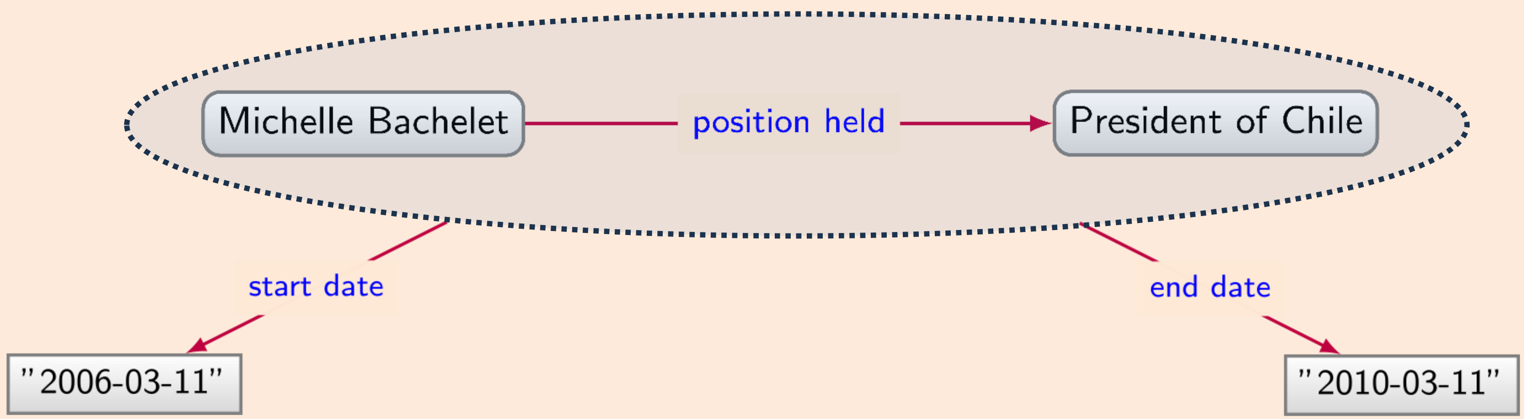
Quotable triples



See [RDF*] for details

Honourable mention: RDF*

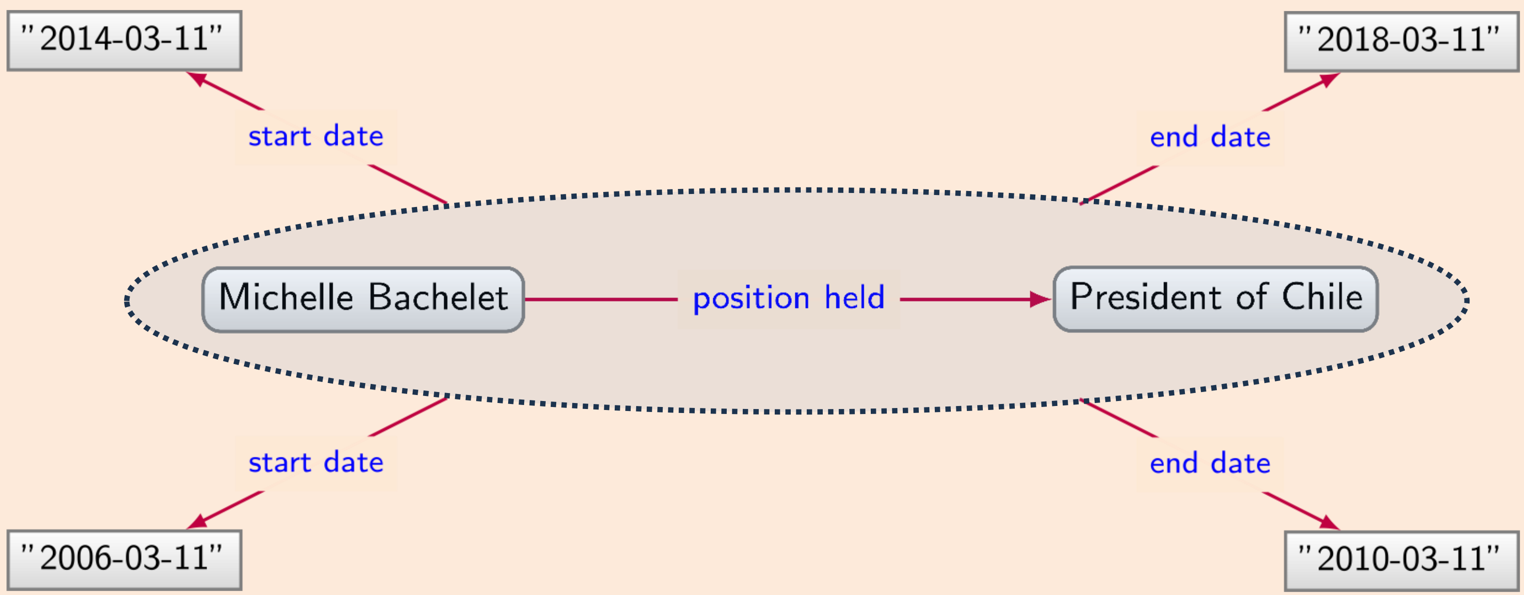
Quotable triples



See [RDF*] for details

Honourable mention: RDF*

Issue: not covering all use cases



See [RDF*] for details

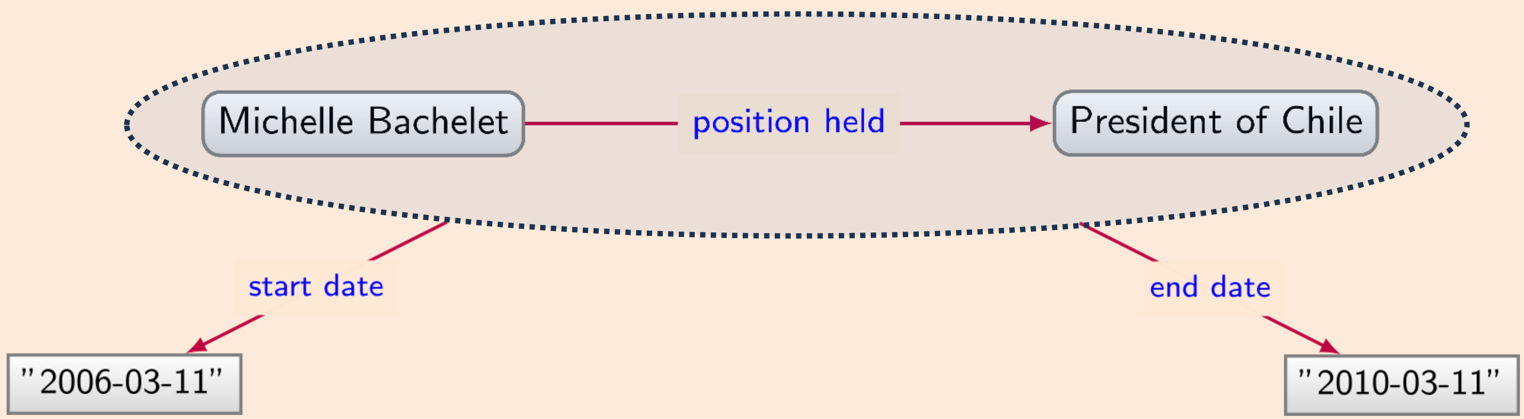
Honourable mention: RDF*

Benefits: neat syntax, being standardized

:Michelle Bachelet :position held :President of Chile .

<<:Michelle Bachelet :position held :President of Chile>> :start date "2006-03-11"^^xsd:date .

<<:Michelle Bachelet :position held :President of Chile>> :end date "2010-03-11"^^xsd:date .

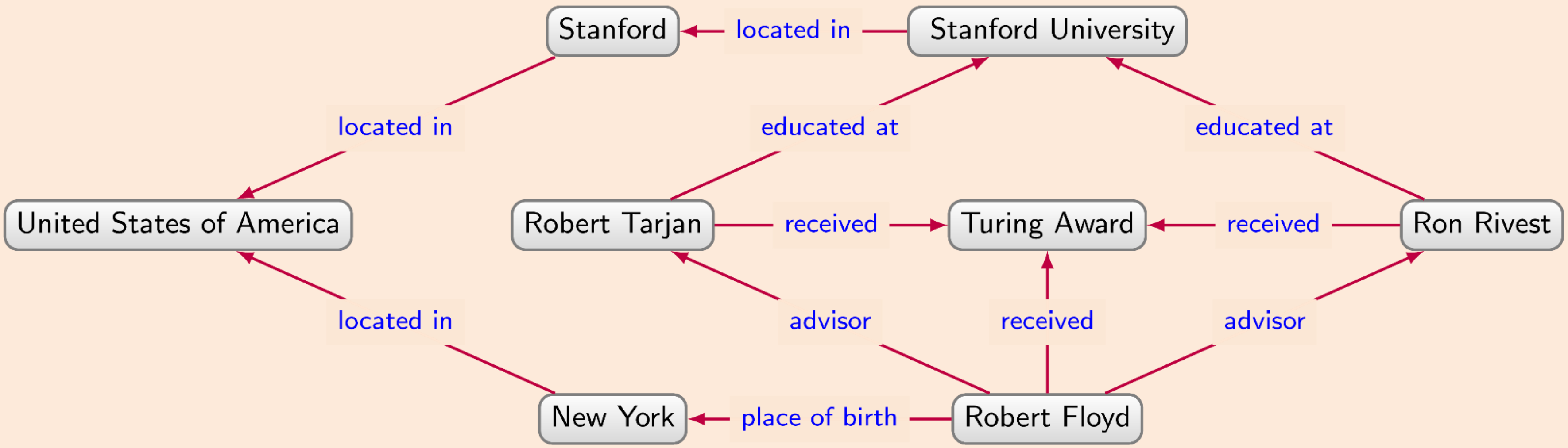


See [RDF*] for details

Graph databases: Why not use relational databases?

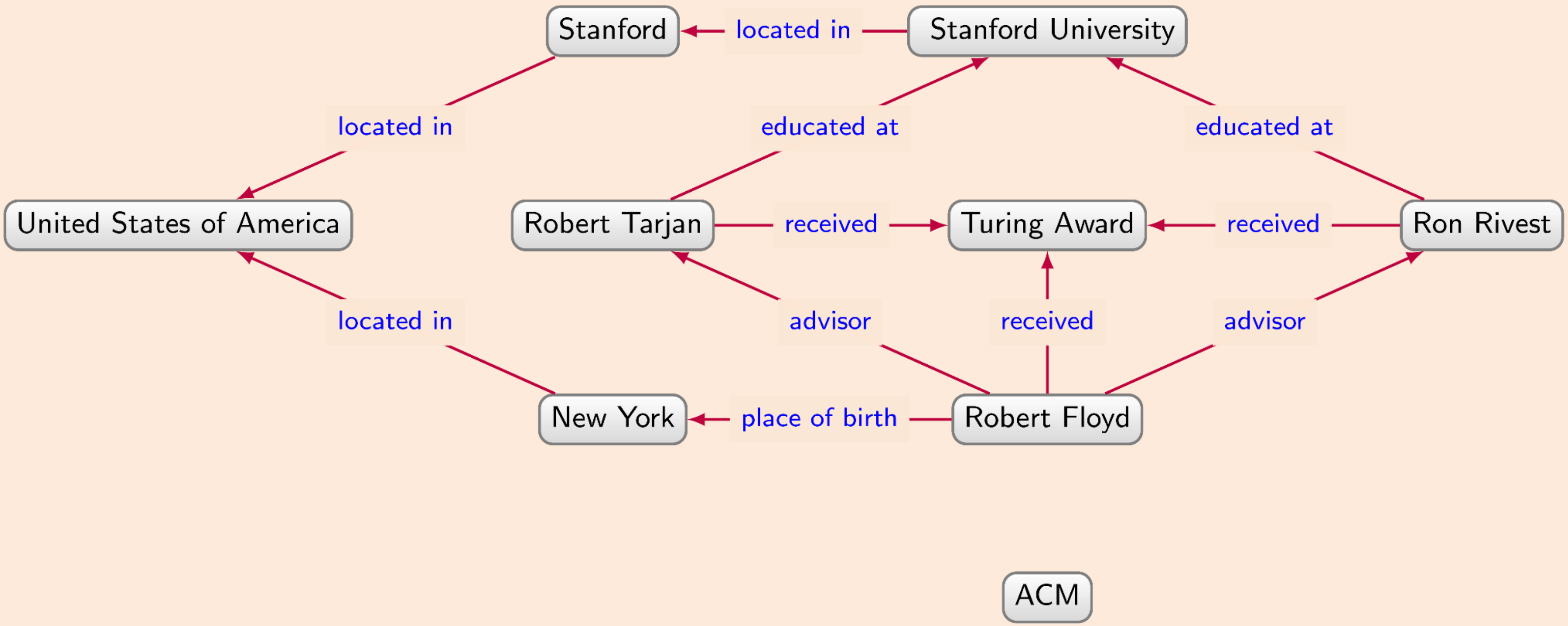
Why use graphs? (flexibility)

RDF



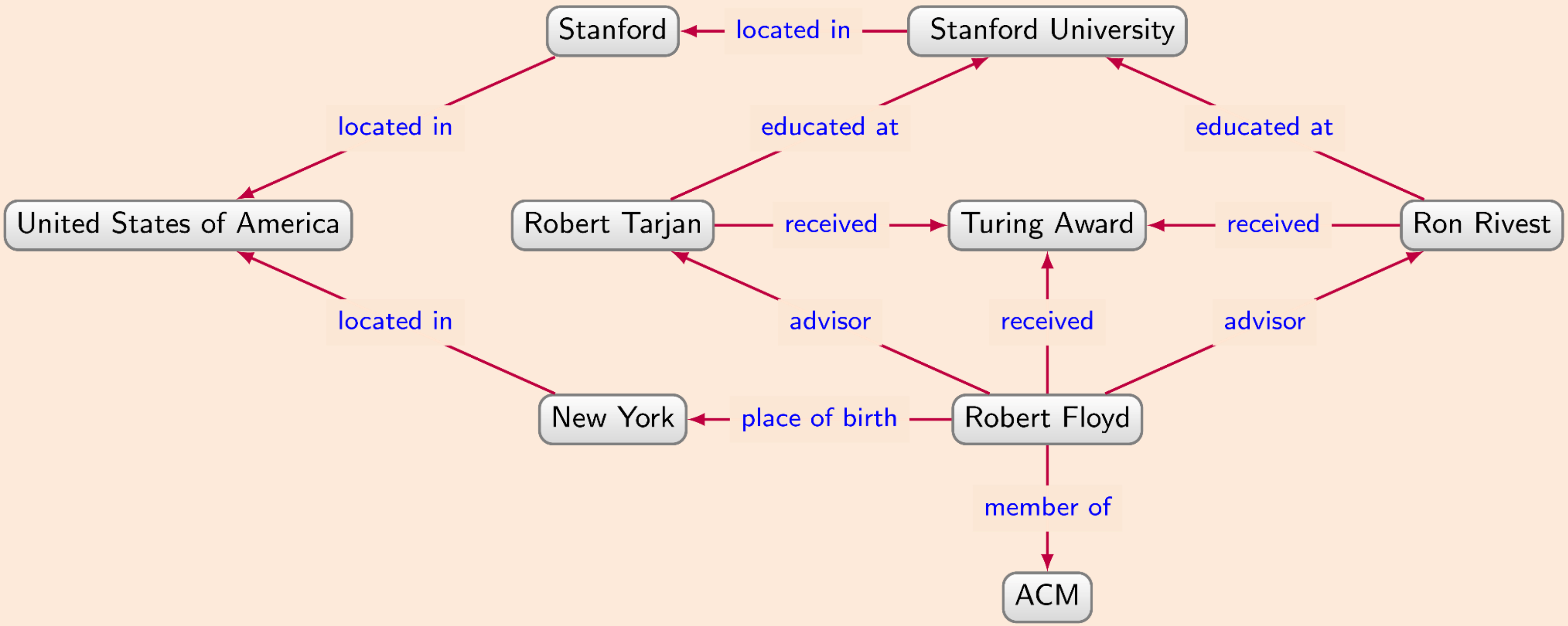
Why use graphs? (flexibility)

RDF



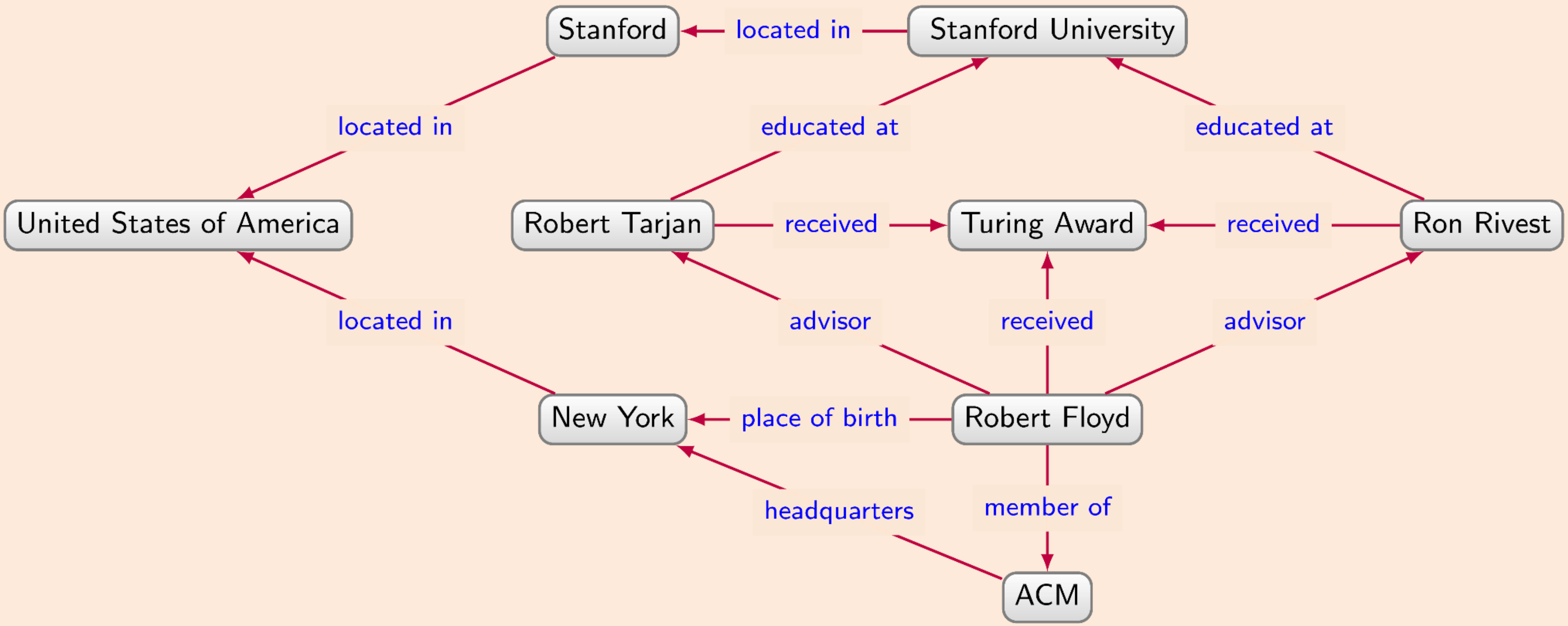
Why use graphs? (flexibility)

RDF



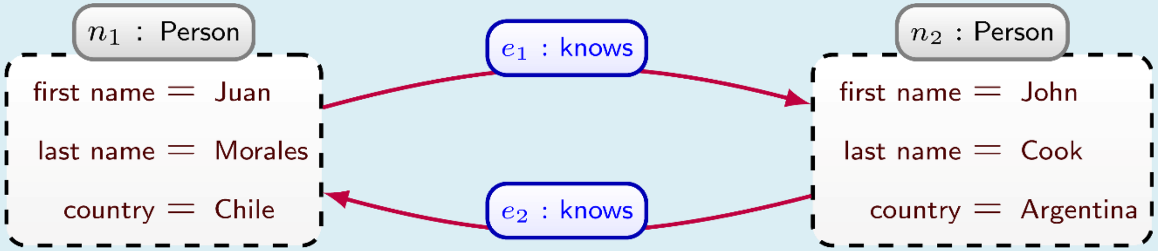
Why use graphs? (flexibility)

RDF



Why use graphs? (flexibility)

Property Graphs

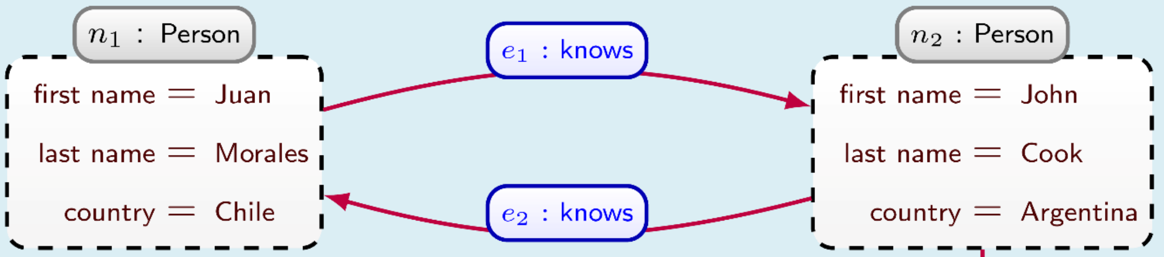


Person

first name	last name	country
Juan	Morales	Chile
John	Cook	Argentina
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...

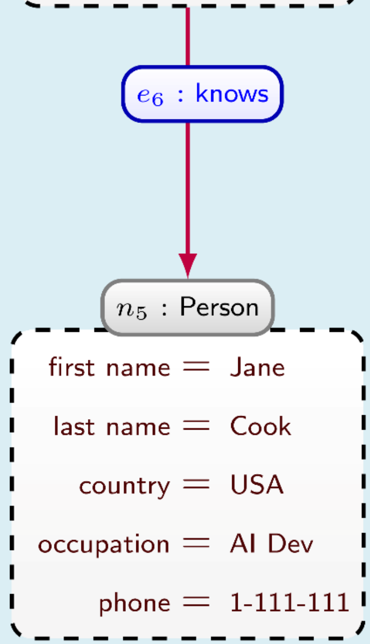
Why use graphs? (flexibility)

Property Graphs



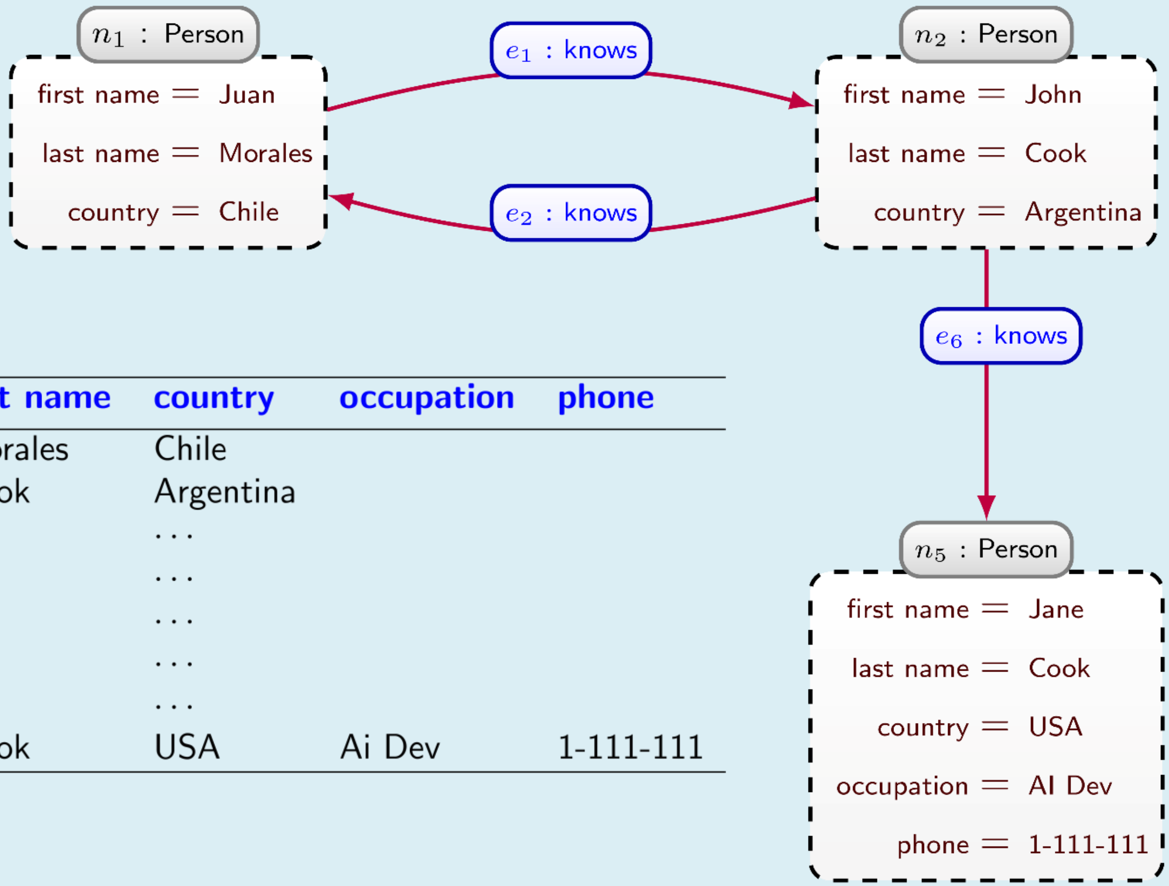
Person

first name	**last name**	**country**
Juan	Morales	Chile
John	Cook	Argentina
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...



Why use graphs? (flexibility)

Property Graphs

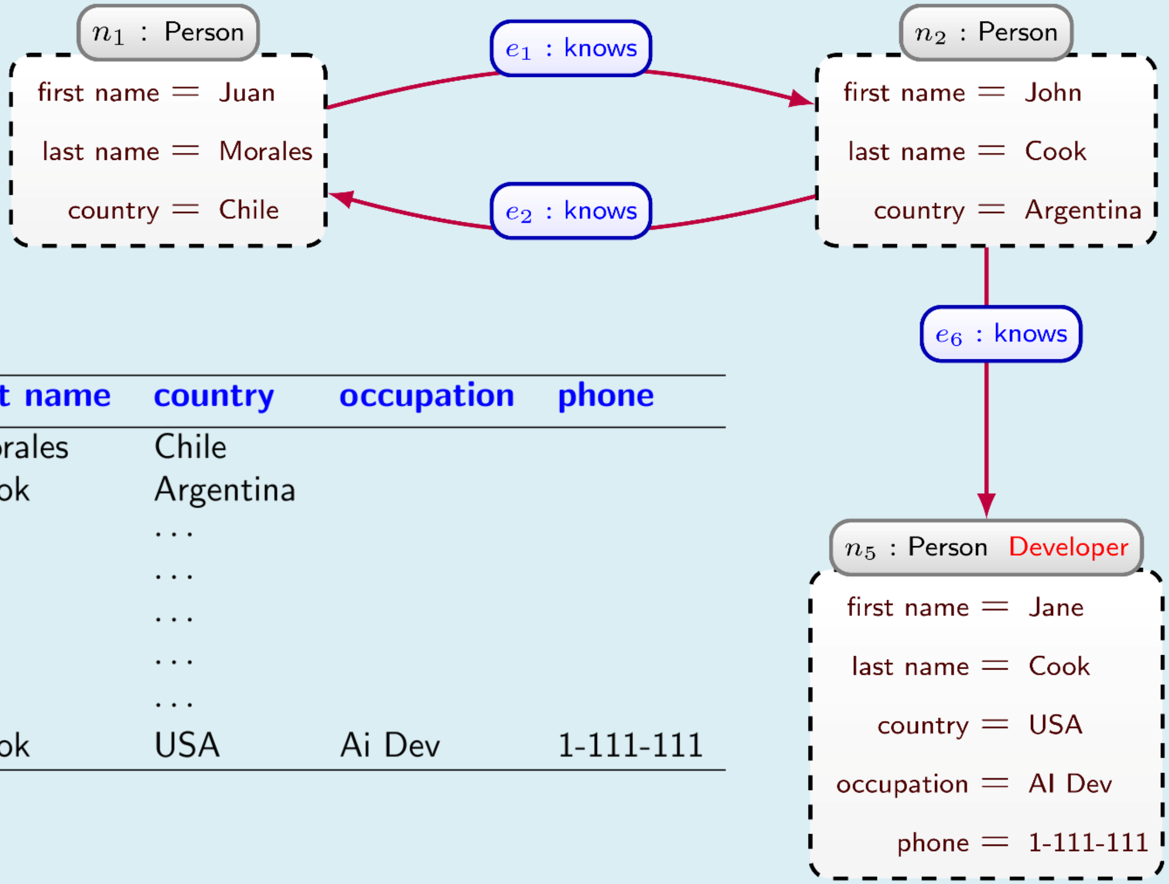


Person

first name	last name	country	occupation	phone
Juan	Morales	Chile		
John	Cook	Argentina		
...	...	...		
...	...	...		
...	...	...		
...	...	...		
...	...	...		
Jane	Cook	USA	Ai Dev	1-111-111

Why use graphs? (flexibility)

Property Graphs



Person

first name	last name	country	occupation	phone
Juan	Morales	Chile		
John	Cook	Argentina		
...	...	...		
...	...	...		
...	...	...		
...	...	...		
...	...	...		
Jane	Cook	USA	Ai Dev	1-111-111

The floor is yours!

Anything you would like to add?

Querying graph databases

Graph query languages

- RDF/edge-labelled graphs:
  - **SPARQL** W3C standard [SPARQL]
  - Bunch of engines (Blazegraph, Jena, Virtuoso, MillenniumDB,...)

- Property graphs:
  - **GQL** fresh ISO standard (very expressive) [GQL22, GQLDigest]
    - Heavily influenced by Neo4J's Cypher [Cypher]
  - **SQL/PGQ**

Graph query languages

Core features of all graph query languages

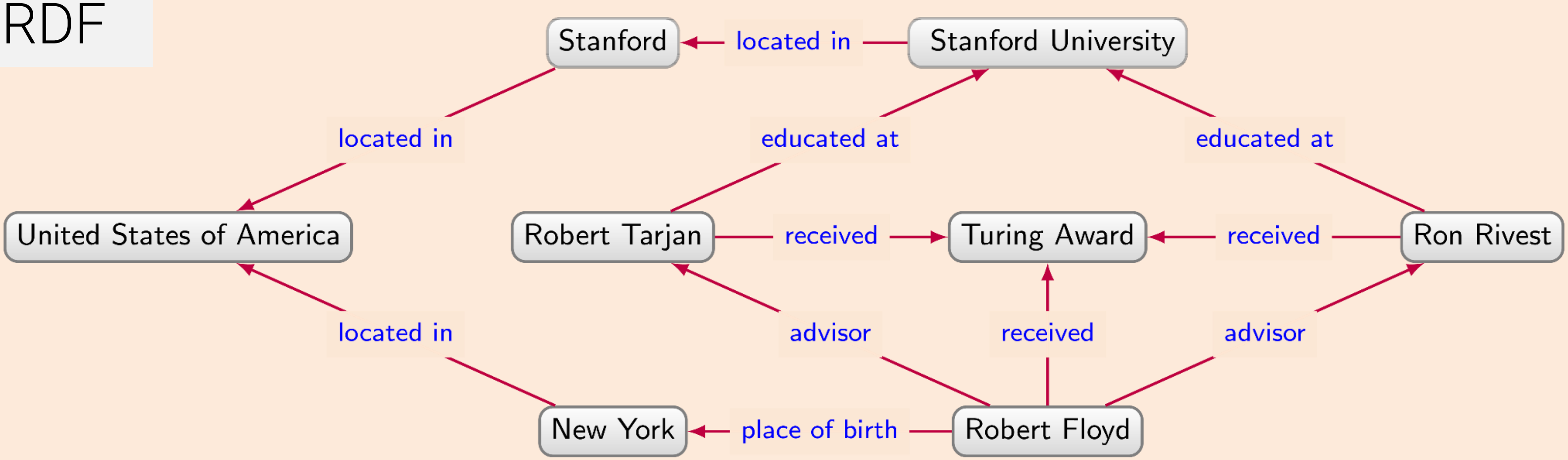
- **Graph patterns**:
  - Find a smaller graph-like pattern in a larger graph
- **Path queries**:
  - Find how the graph nodes are connected via paths
- Navigational graph patterns:
  - Put path queries into graph patterns
- Complex graph queries:
  - Filters, aggregation, union, projection, selection, ...

See [AABHRV17] for details

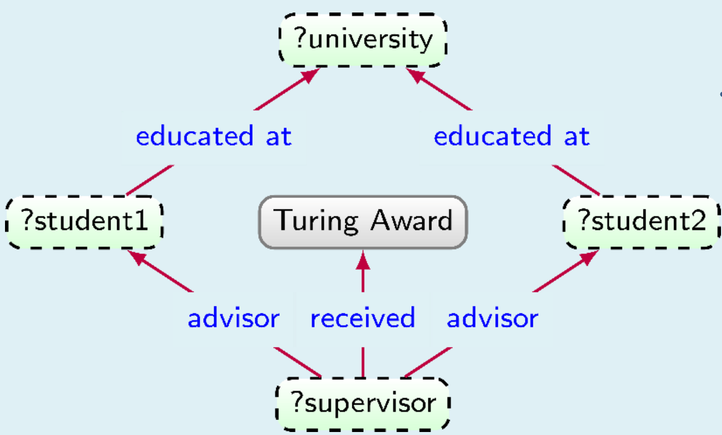
Graph Patterns

Basic graph patterns

RDF



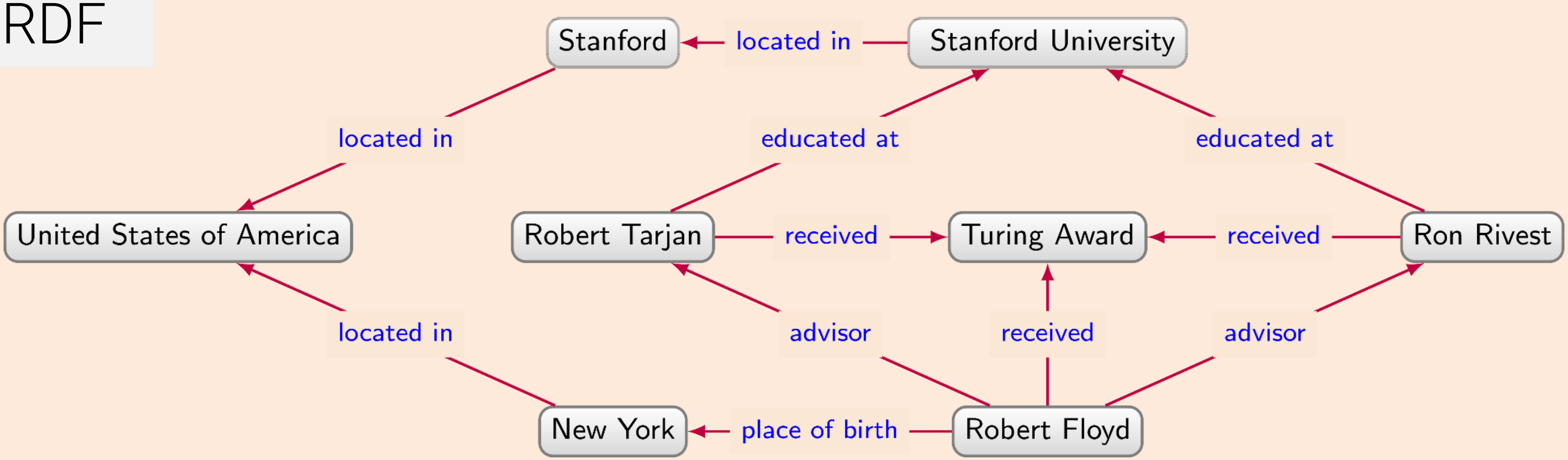
Academic siblings whose supervisor won the Turing Award



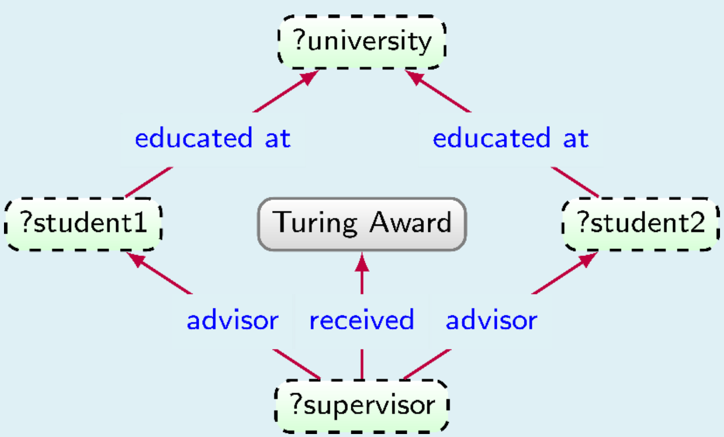
Idea:
Match this into the main graph (preserve constants)

Basic graph patterns

RDF



Academic siblings whose supervisor won the Turing Award

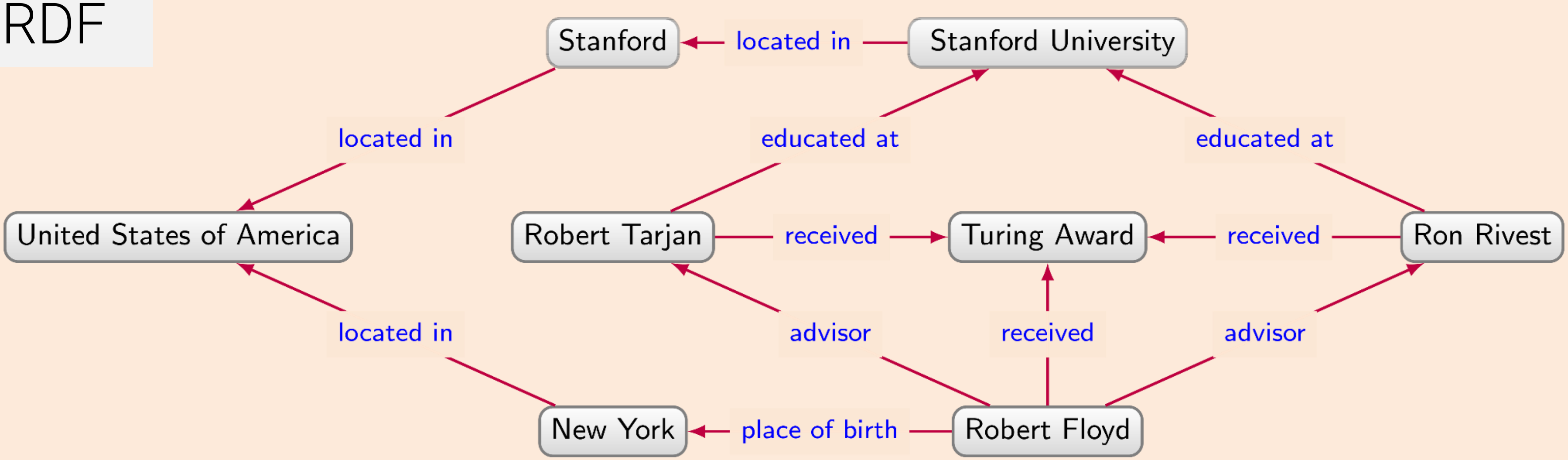


Semantics: Homomorphism

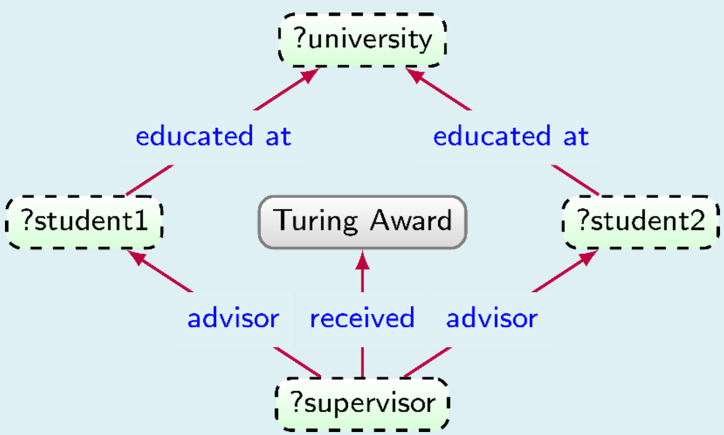
?supervisor	?student1	?student2	?univeristy
Robert Floyd	Robert Tarjan	Ron Rivest	Stanford Univeritsy
Robert Floyd	Ron Rivest	Robert Tarjan	Stanford Univeritsy
Robert Floyd	Robert Tarjan	Robert Tarjan	Stanford Univeritsy
Robert Floyd	Ron Rivest	Ron Rivest	Stanford Univeritsy

Basic graph patterns

RDF



Academic siblings whose supervisor won the Turing Award

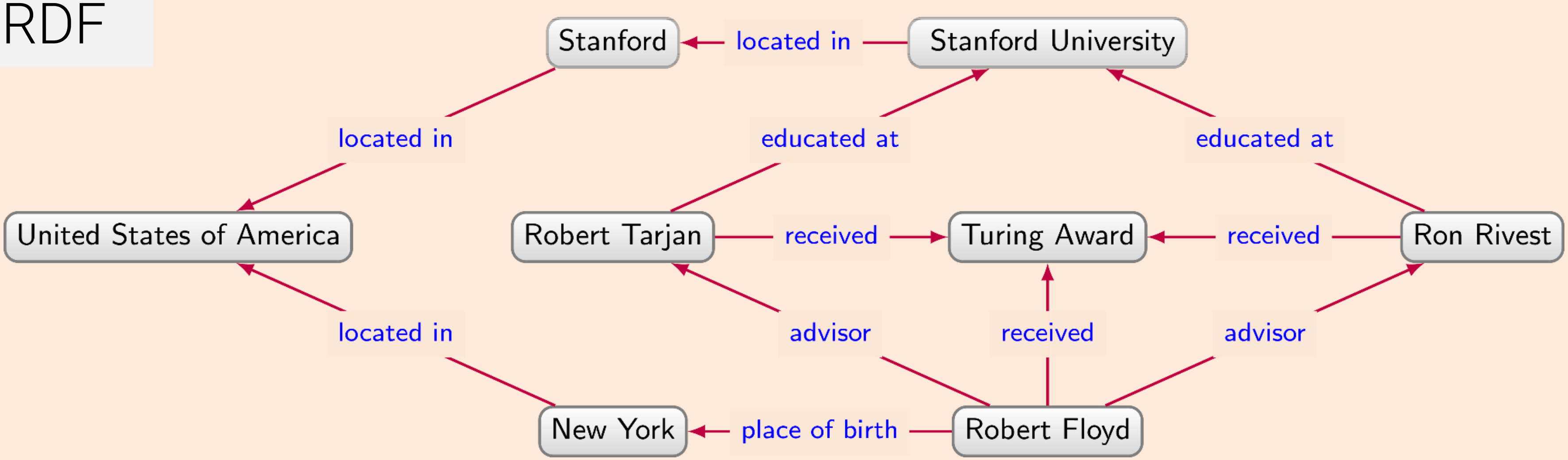


Semantics: Isomorphism

?supervisor	?student1	?student2	?univeristy
Robert Floyd	Robert Tarjan	Ron Rivest	Stanford Univeritsy
Robert Floyd	Ron Rivest	Robert Tarjan	Stanford Univeritsy
Robert Floyd	Robert Tarjan	Robert Tarjan	Stanford Univeritsy
Robert Floyd	Ron Rivest	Ron Rivest	Stanford Univeritsy

Basic graph patterns

RDF

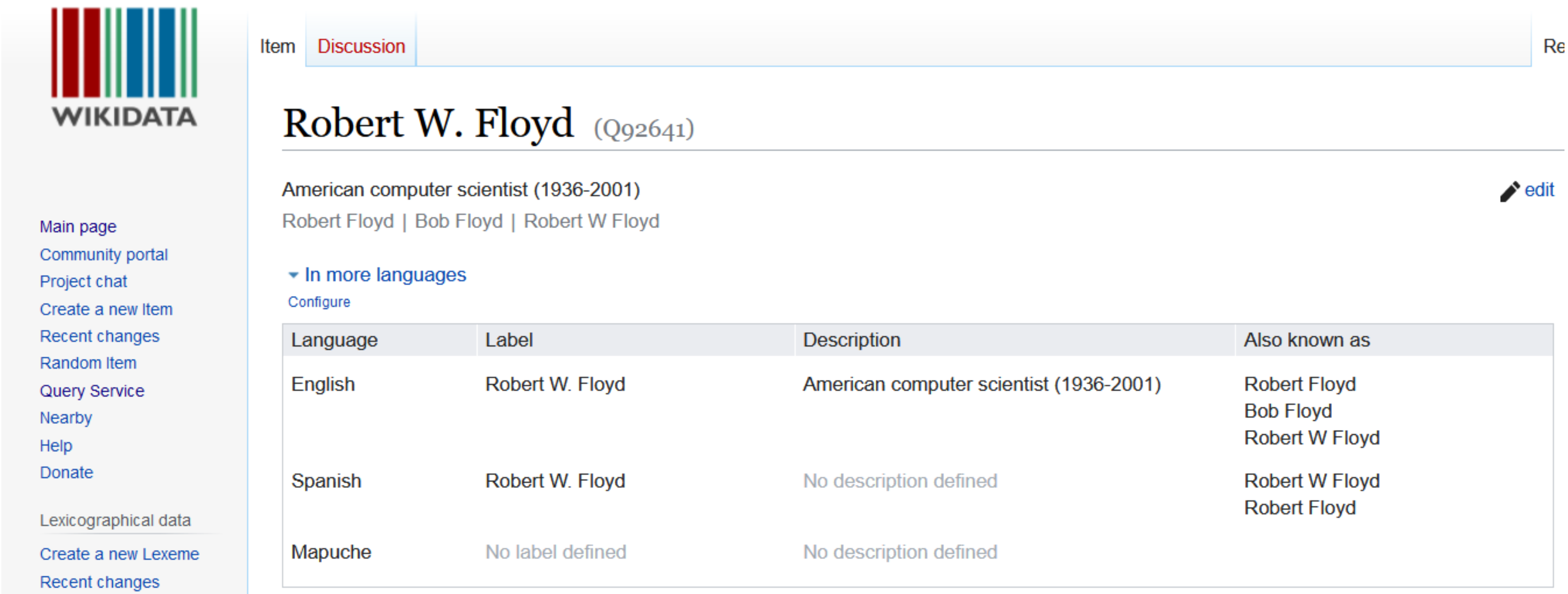


Support in RDF databases

SPARQL:

- Known as **triple patterns** [PAG09]
- Basically joins over the Edge(src,label,tgt) table

Let’s see this on Wikidata/SPARQL



WIKIDATA

Item | Discussion | Re

Main page
Community portal
Project chat
Create a new Item
Recent changes
Random Item
Query Service
Nearby
Help
Donate

Lexicographical data

Create a new Lexeme
Recent changes

Robert W. Floyd (Q92641)

American computer scientist (1936-2001)

edit

Robert Floyd | Bob Floyd | Robert W Floyd

In more languages

Configure

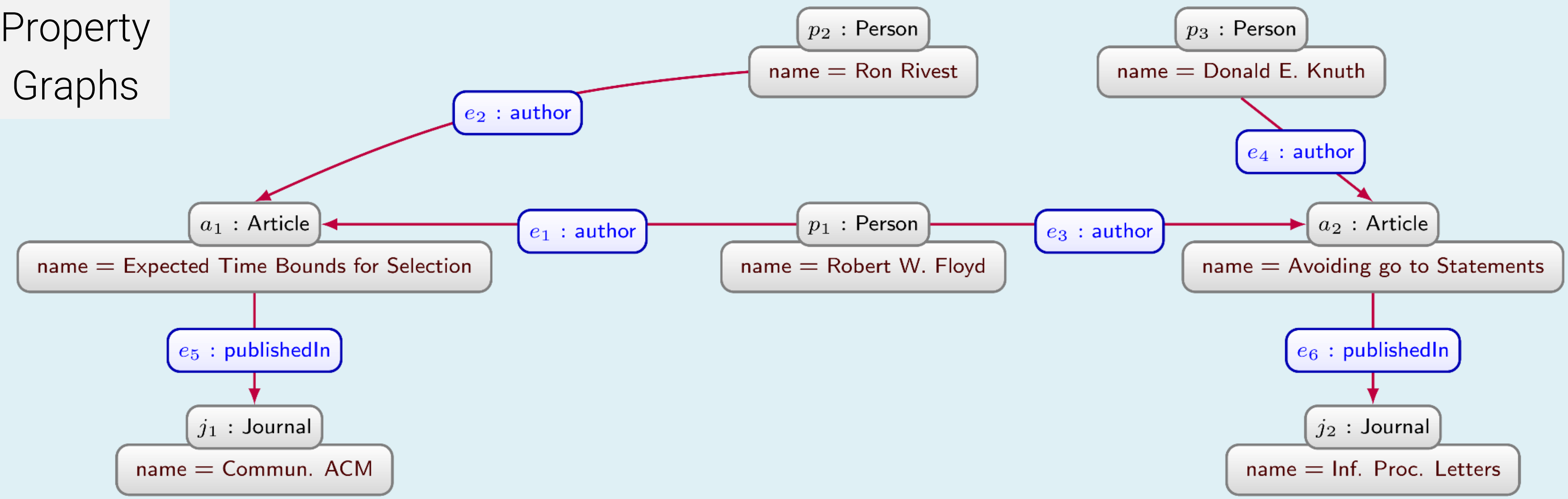
Language	Label	Description	Also known as
English	Robert W. Floyd	American computer scientist (1936-2001)	Robert Floyd Bob Floyd Robert W Floyd
Spanish	Robert W. Floyd	No description defined	Robert W Floyd Robert Floyd
Mapuche	No label defined	No description defined	

https://wikidata.imfd.cl

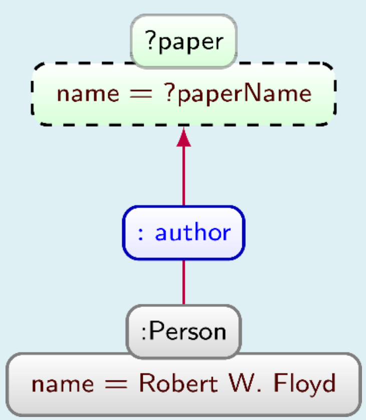
Query1 Query2 Query3

Basic graph patterns

Property Graphs



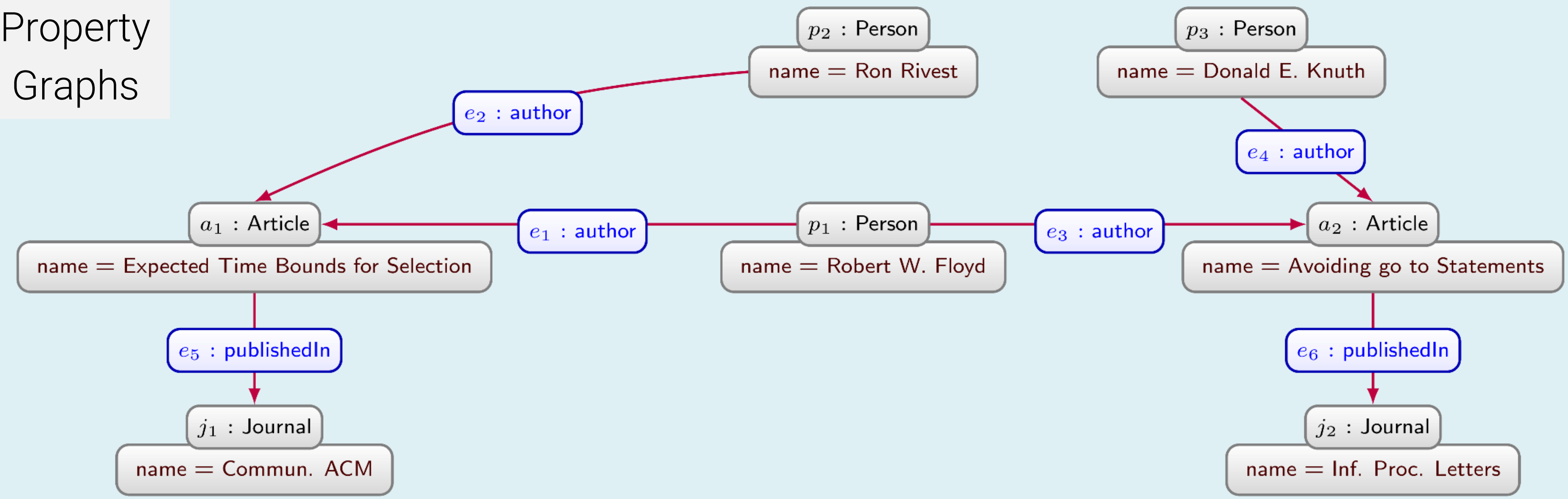
Papers written by Robert Floyd



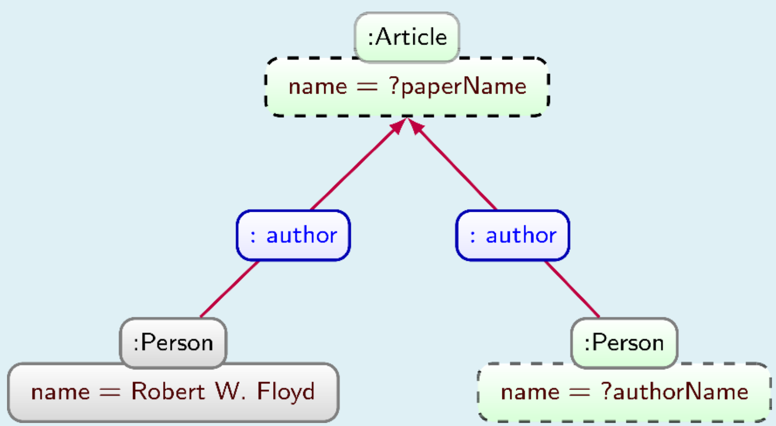
?paperName	?paper
Expected Time Bounds for Selection	a_1
Note on Avoiding go to Statements	a_2

Basic graph patterns

Property Graphs



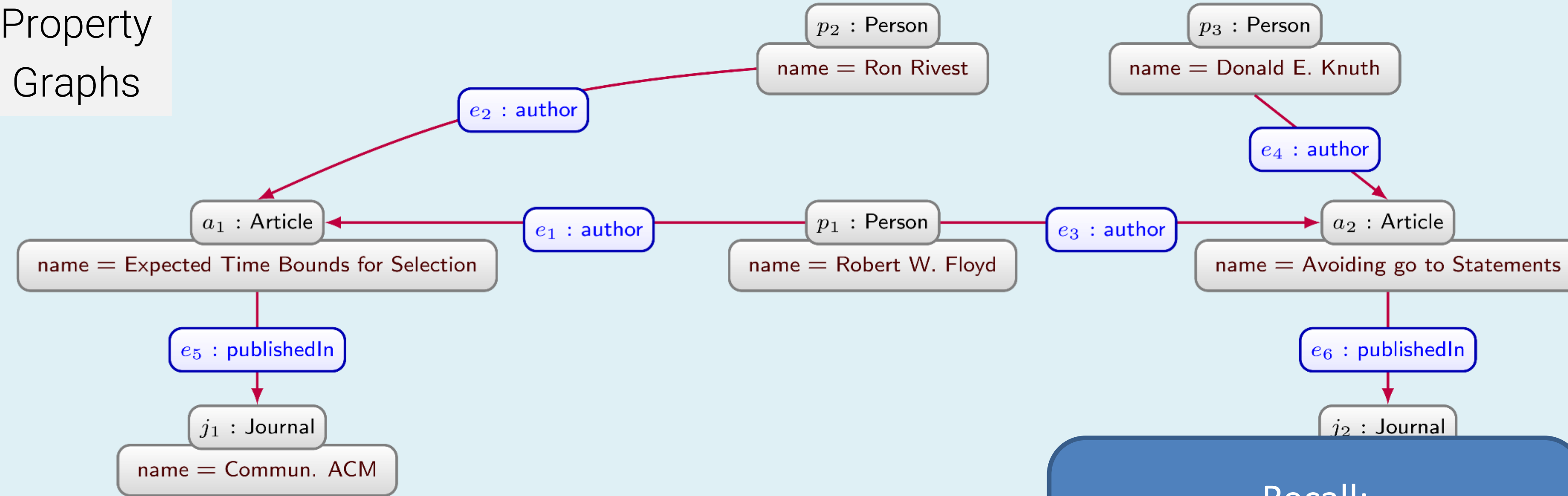
Co-authors of Robert Floyd



?authorName	?paperName
Ron Rivest	Expected Time Bounds for Selection
Donald E. Knuth	Note on Avoiding go to Statements
Robert W. Floyd	Expected Time Bounds for Selection
Robert W. Floyd	Note on Avoiding go to Statements

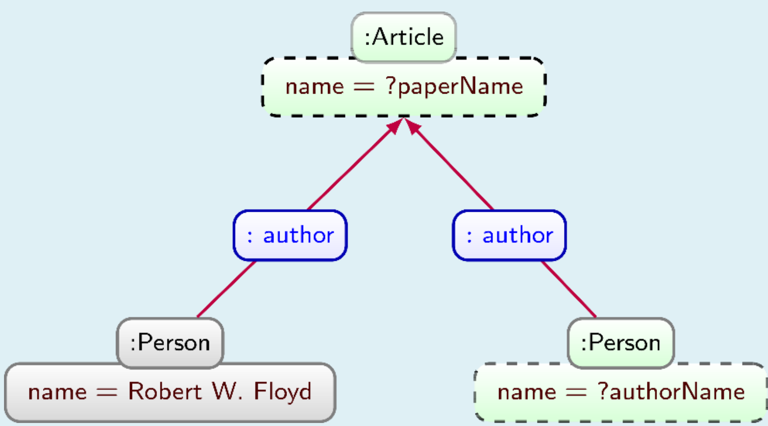
Basic graph patterns

Property Graphs



Co-authors of Robert Fl[illegible]

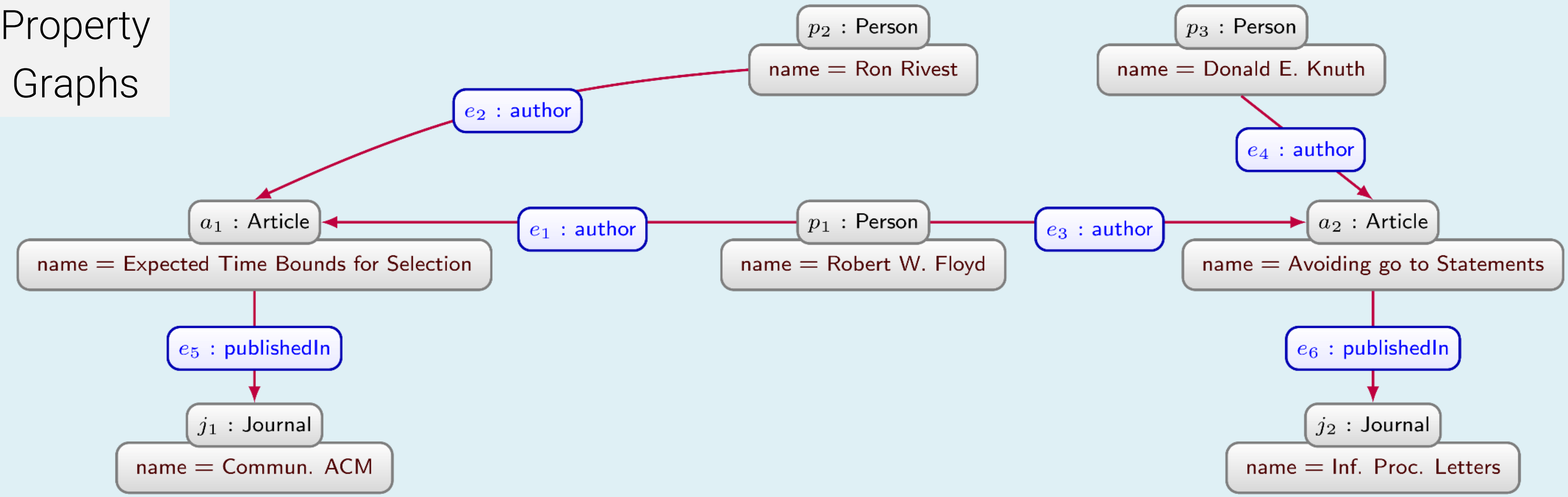
Recall:
Match the pattern into the graph and nothing else!



?authorName	?[illegible]ame
Ron Rivest	[illegible]ected Time Bounds for Selection
Donald E. Knuth	Note on Avoiding go to Statements
Robert W. Floyd	Expected Time Bounds for Selection
Robert W. Floyd	Note on Avoiding go to Statements

Basic graph patterns

Property Graphs



Support in property graph databases

GQL:

- Similar as in SPARQL [GQLDigest, GQL]
- But now we have more things to consider
  - Labels, attribute values, etc.

Let’s see this on BibKG/GQL

BibKG QUERY DOCS

```
// Papers by Robert W. Floyd
MATCH (?x {name:"Robert W. Floyd"})-[?p :author_of]->(?y)
RETURN ?y, ?y.name
```

EXAMPLES RUN

EXPORT AS CSV

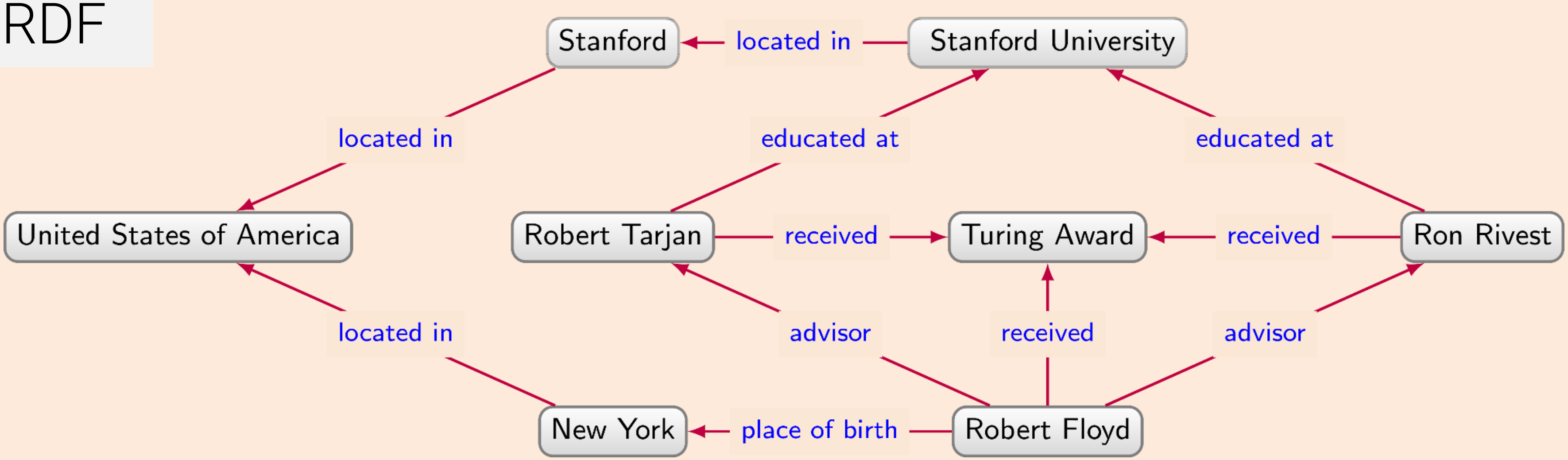
y	y.name
j_jacm_FloydU82	"The Compilation of Regular Expressions into Integrated Circuits."
j_cacm_Floyd62a	"Algorithm 97: Shortest path."

https://bibkg.imfd.cl

Path Queries

Regular path queries

RDF



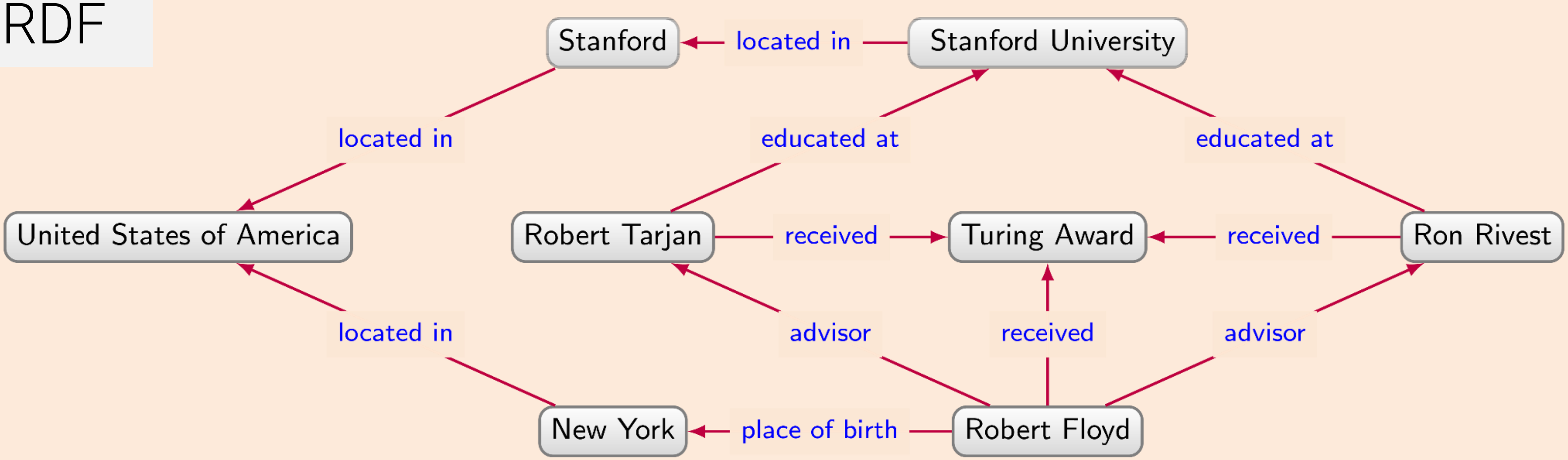
A generic RPQ

Idea:

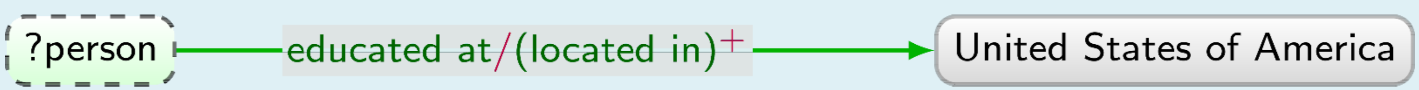
- find pairs of nodes
- connected by a path
- whose edge labels are a word matching regex

Regular path queries

RDF



People educated at a university in the USA

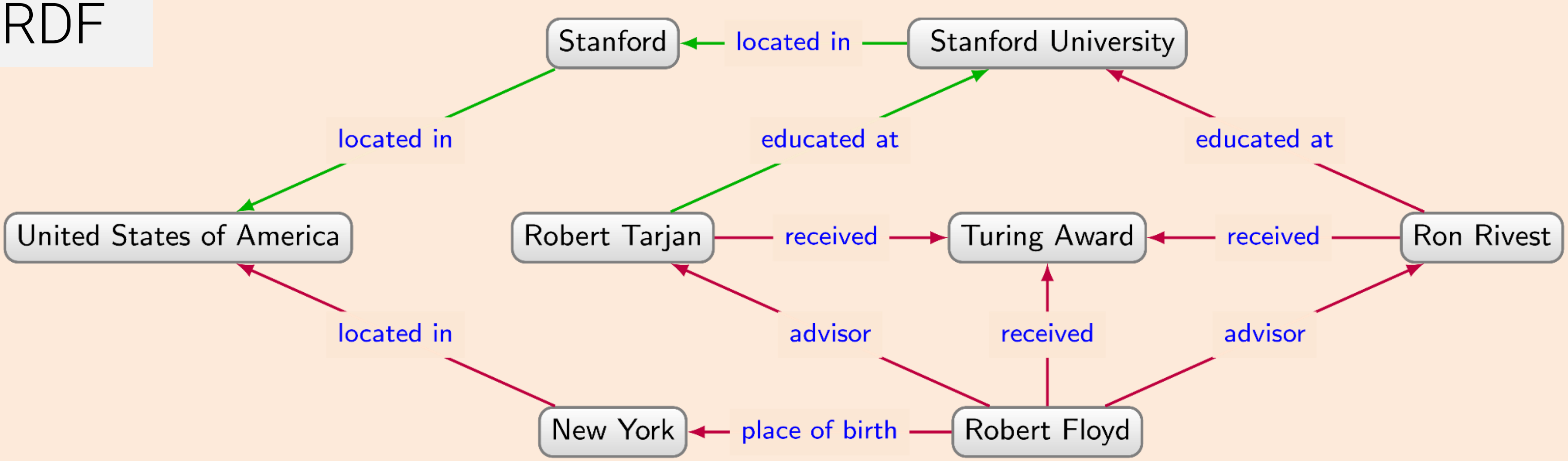


Idea:

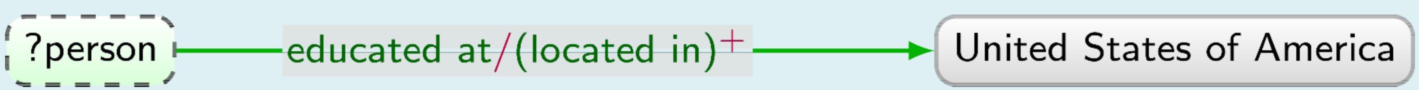
- traverse an educated at-labelled edge
- then any number of located in-labelled edges
- until you reach the node "United States of America"

Regular path queries

RDF



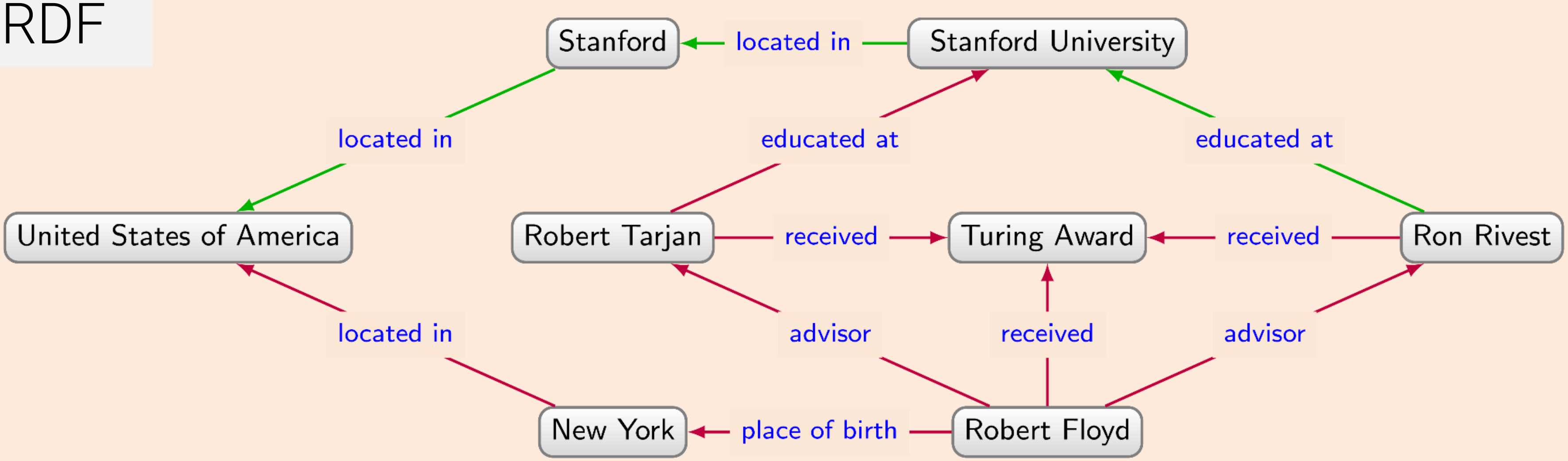
People educated at a university in the USA



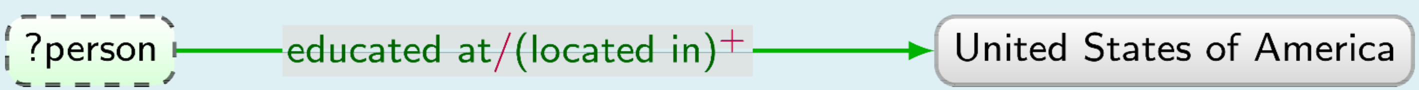
?person
Robert Tarjan
Ron Rivest

Regular path queries

RDF



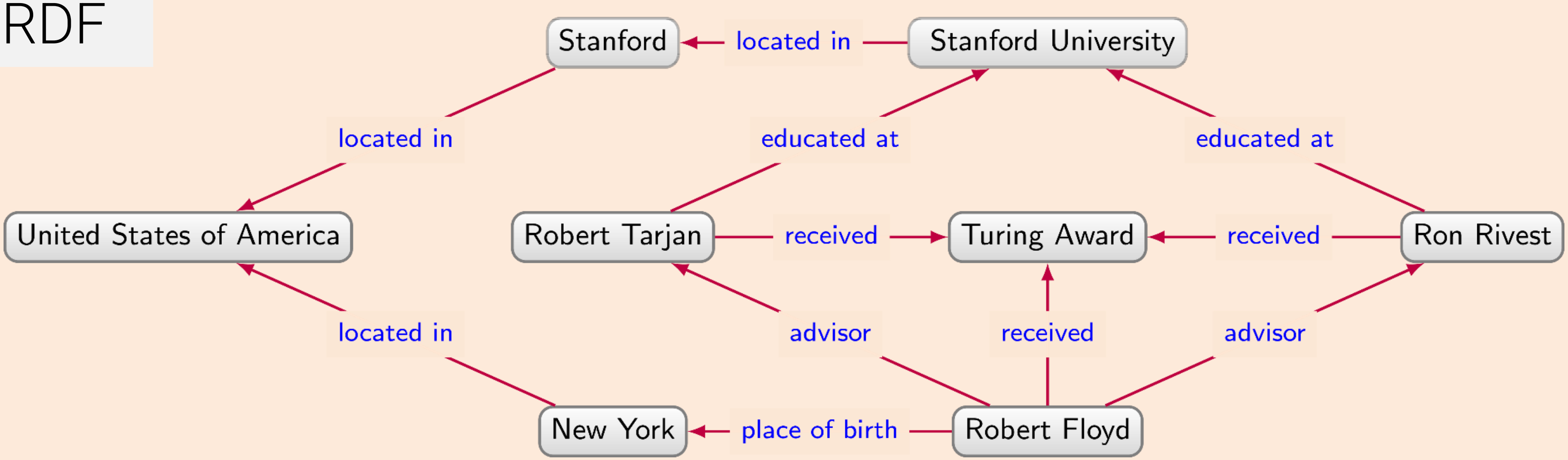
People educated at a university in the USA



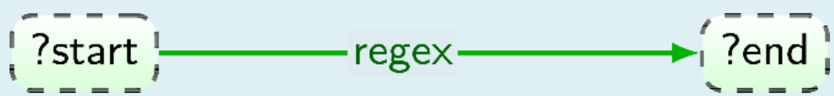
?person
Robert Tarjan
Ron Rivest

Regular path queries

RDF



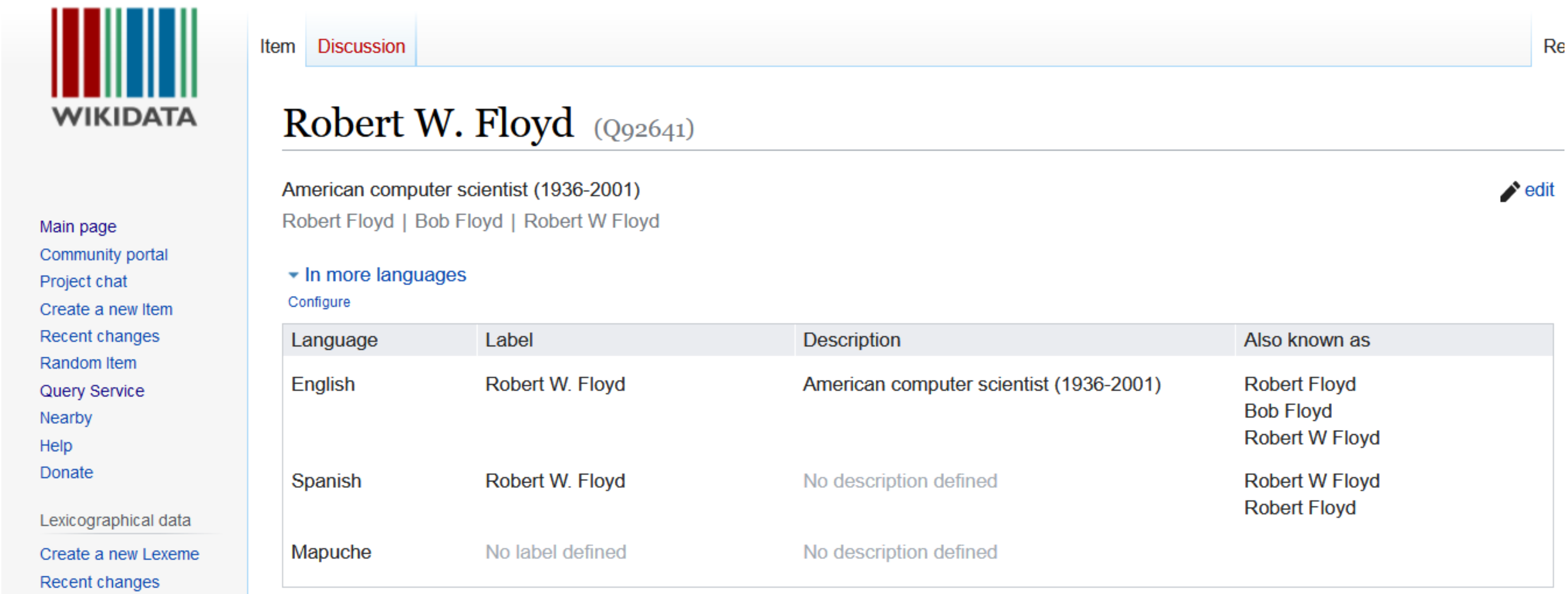
A generic RPQ



SPARQL:

- Known as **property paths** [KRRV15]
- Based on 2-way regular path queries (RPQs) [2RPQs, MW95]
- Essentially a reachability check – no path is returned

Let’s see this on Wikidata/SPARQL

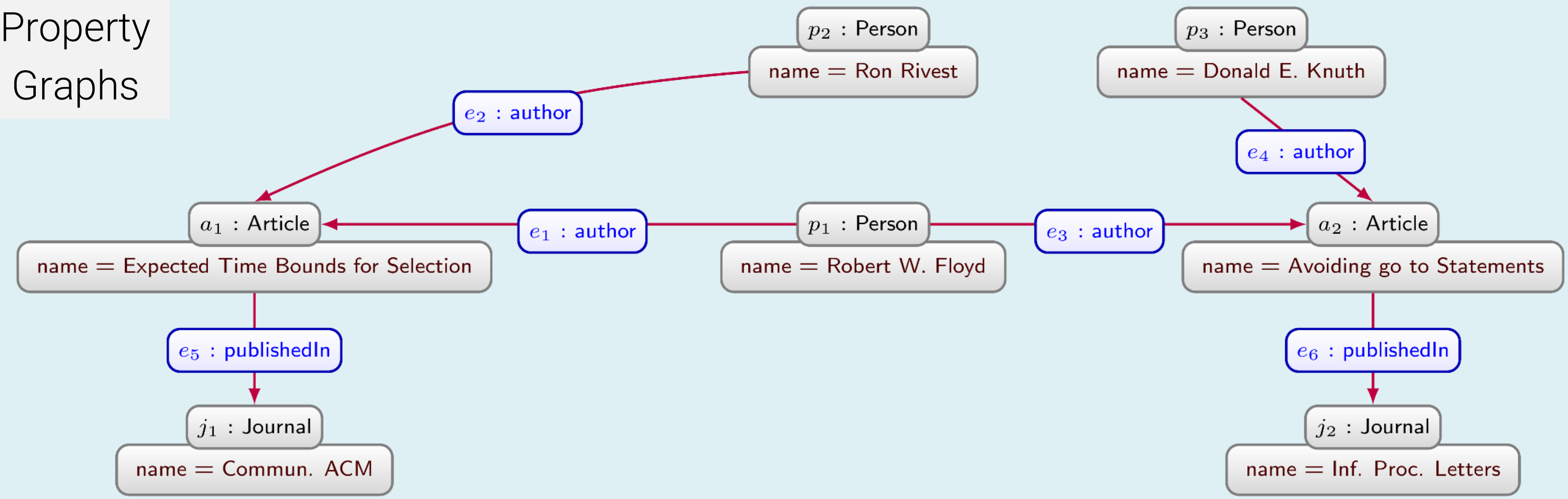


https://wikidata.imfd.cl

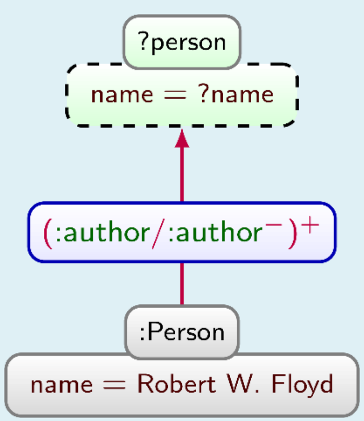
[Query]()

Regular path queries – but extended

Property Graphs

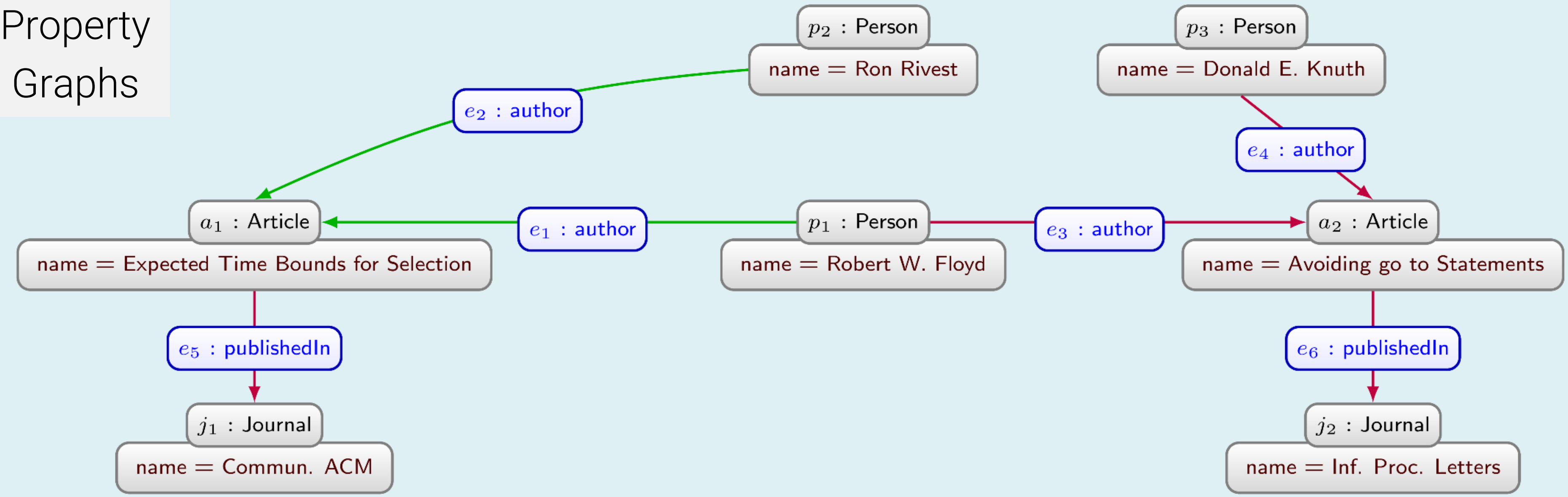


People with a finite Floyd number

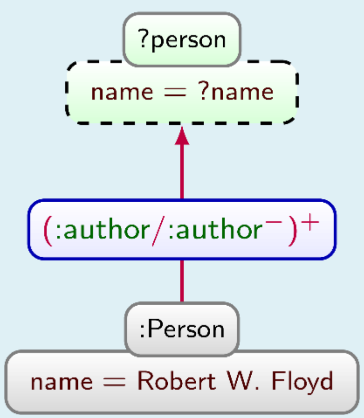


Regular path queries – but extended

Property Graphs



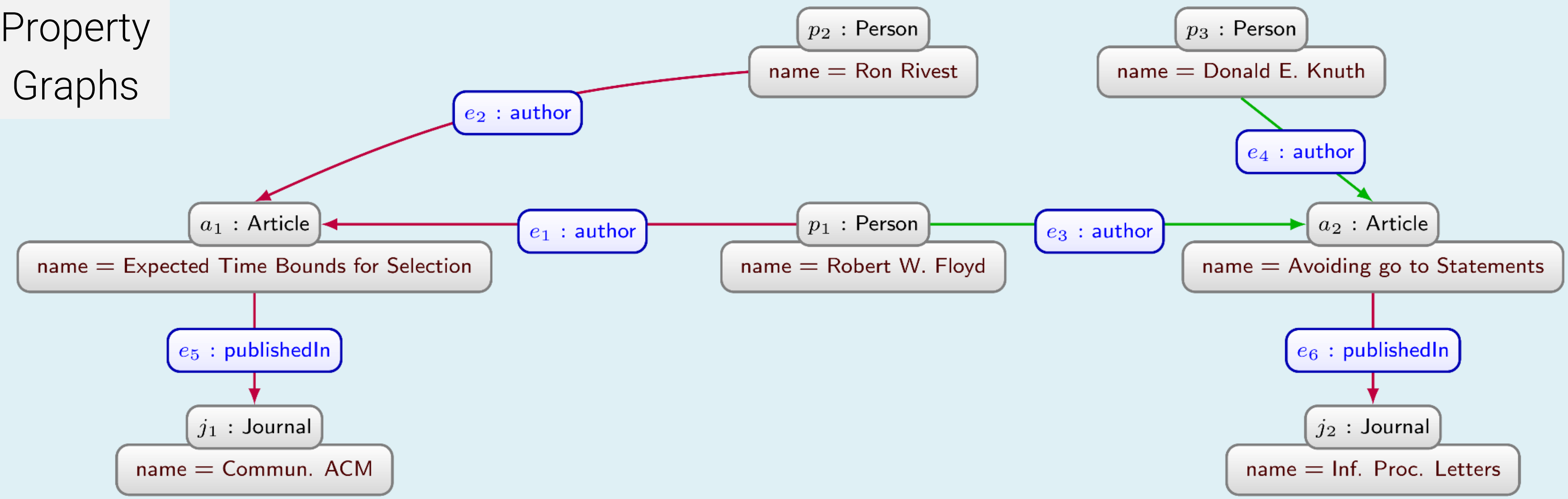
People with a finite Floyd number



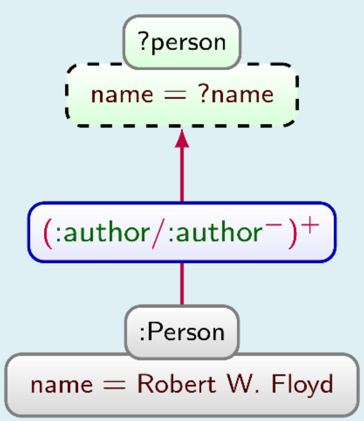
?name	?person
Ron Rivest	p_2
Donald E. Knuth	p_3
Robert W. Floyd	p_1

Regular path queries – but extended

Property Graphs



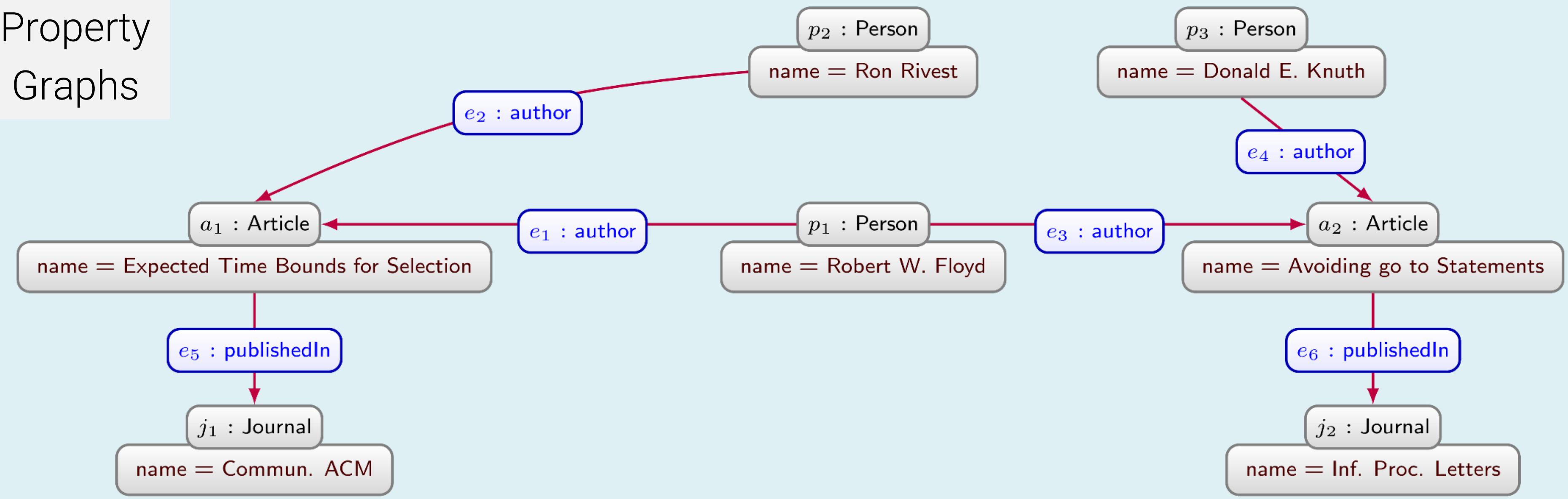
People with a finite Floyd number



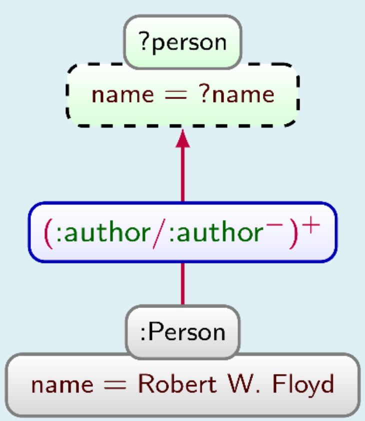
?name	?person
Ron Rivest	p_2
Donald E. Knuth	p_3
Robert W. Floyd	p_1

Regular path queries – but extended

Property Graphs



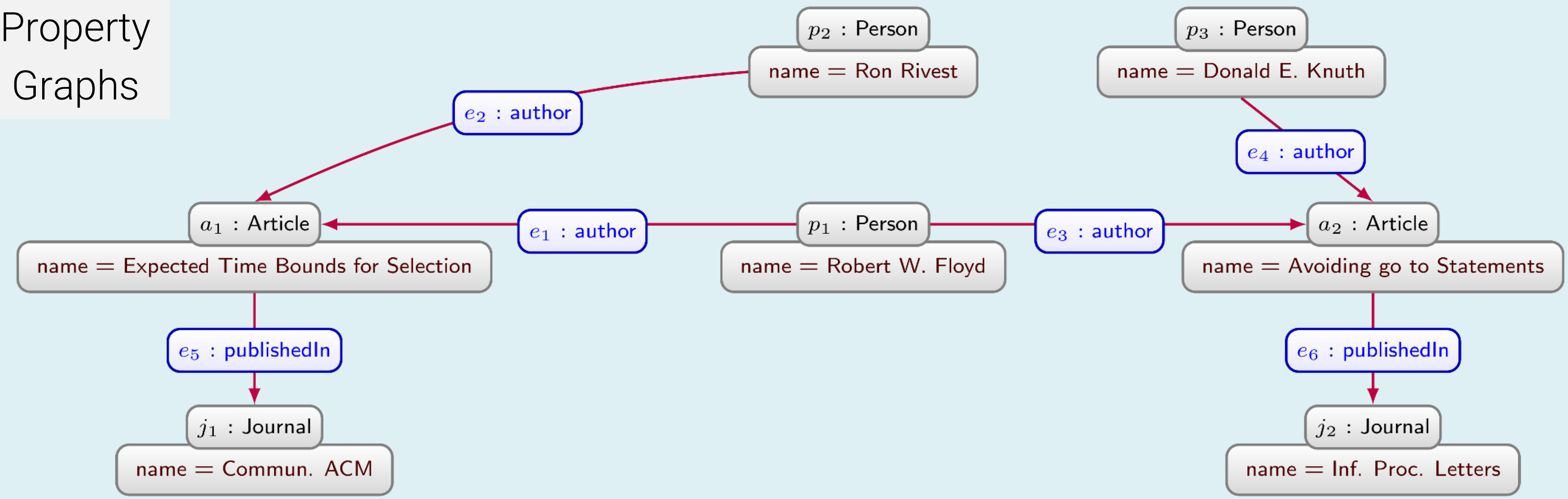
People with a finite Floyd number



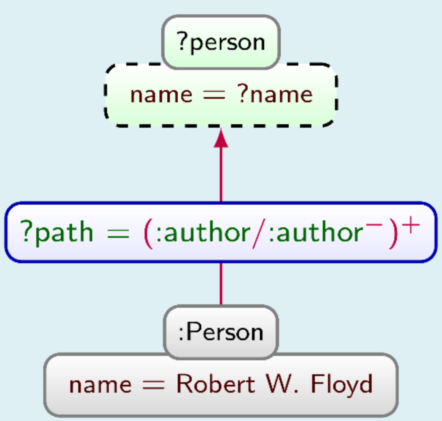
?name	?person
Ron Rivest	p_2
Donald E. Knuth	p_3
Robert W. Floyd	p_1

Regular path queries – but extended

Property Graphs



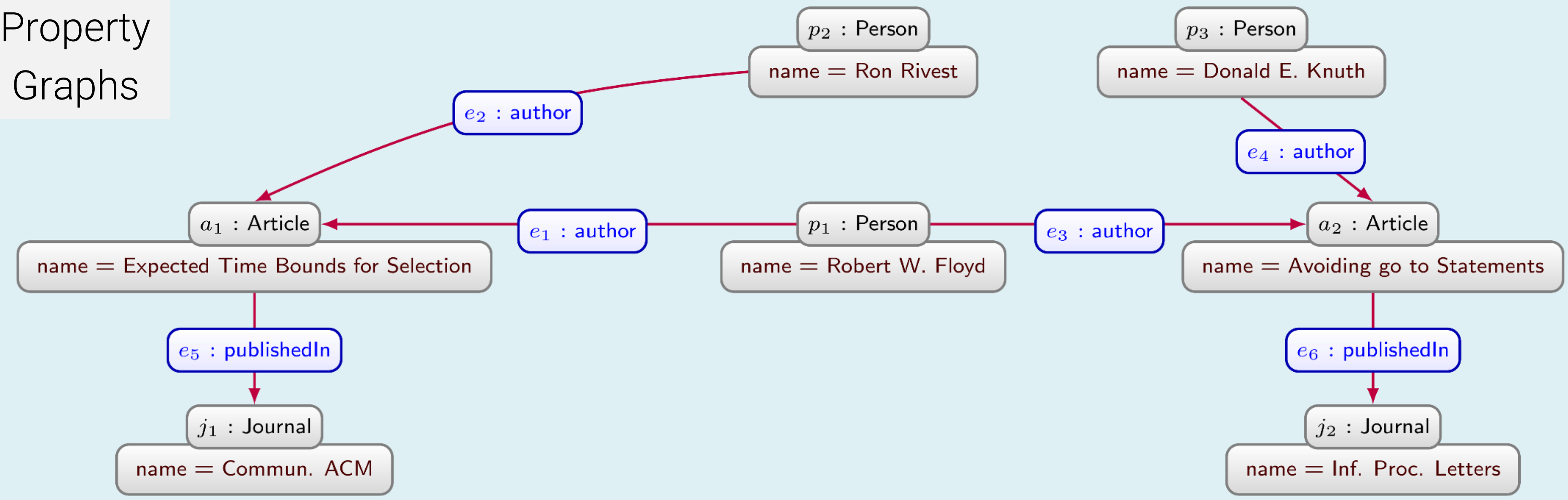
People with a finite Floyd number – and a path to them



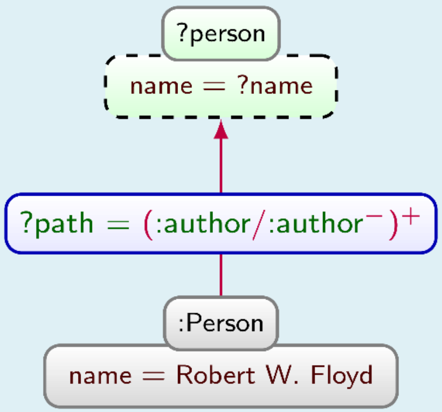
?name	?person	?path
Ron Rivest	p_2	$p_1 \to a_1 \leftarrow p_2$
Donald E. Knuth	p_3	$p_1 \to a_2 \leftarrow p_3$
Robert W. Floyd	p_1	$p_1 \to a_1 \leftarrow p_1$
Robert W. Floyd	p_1	$p_1 \to a_2 \leftarrow p_1$

Regular path queries – but extended

Property Graphs



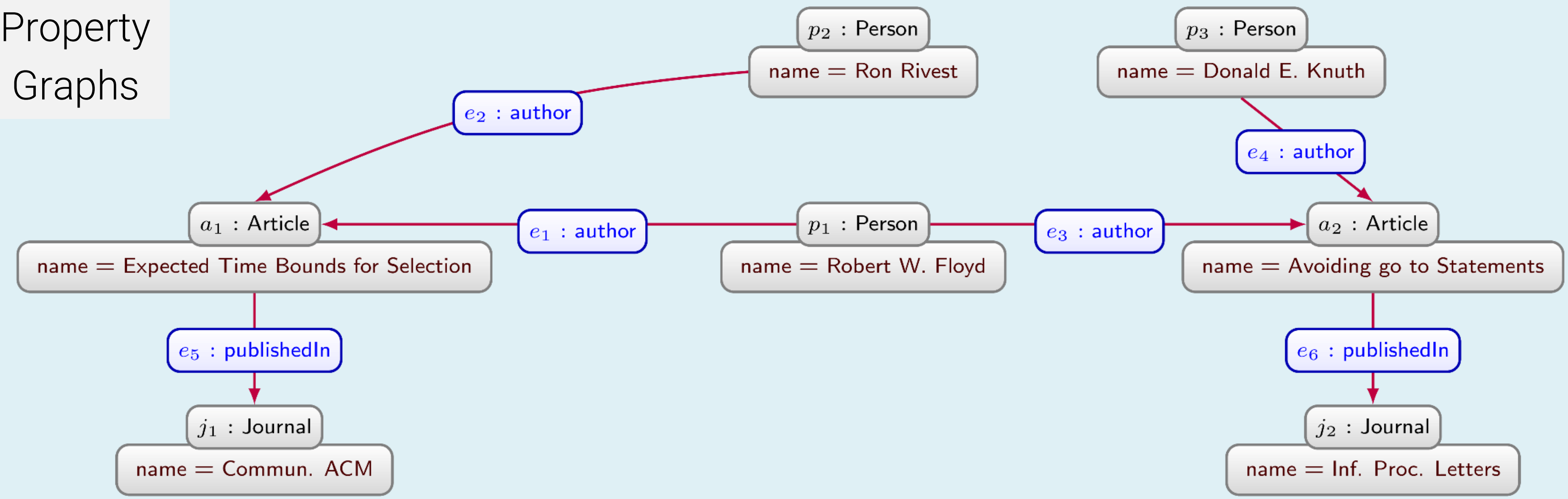
People with a finite Floyd number – and a path to them



?name	?person	?path
Ron Rivest	p_2	$p_1 \to a_1 \leftarrow p_2$
Donald E. Knuth	p_3	$p_1 \to a_2 \leftarrow p_3$
Robert W. Floyd	p_1	$p_1 \to a_1 \leftarrow p_1$
Robert W. Floyd	p_1	$p_1 \to a_2 \leftarrow p_1$

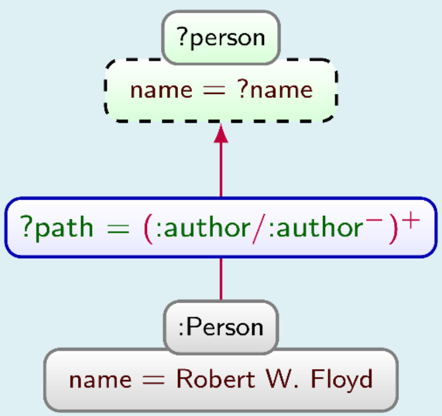
Regular path queries – but extended

Property Graphs



People with a finite Floyd number – and

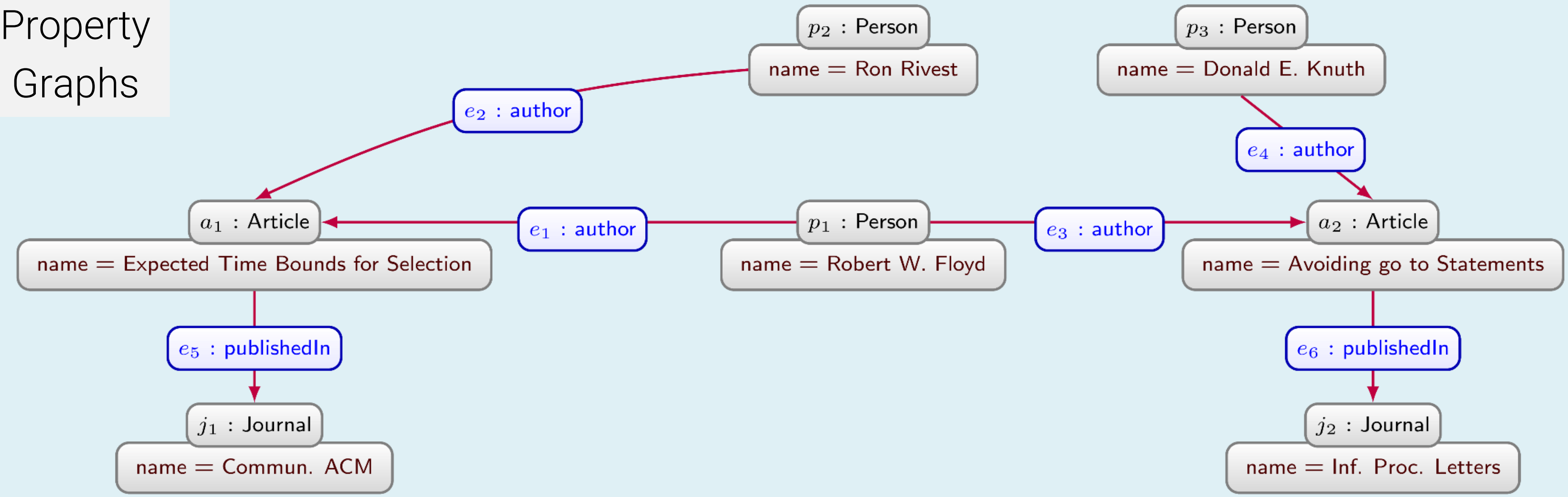
Which paths?
More on this soon!



?name	?[illegible]n	?path
Ron Rivest	p_2	$p_1 \to a_1 \leftarrow p_2$
Donald E. Knuth	p_3	$p_1 \to a_2 \leftarrow p_3$
Robert W. Floyd	p_1	$p_1 \to a_1 \leftarrow p_1$
Robert W. Floyd	p_1	$p_1 \to a_2 \leftarrow p_1$

Regular path queries – but extended

Property Graphs

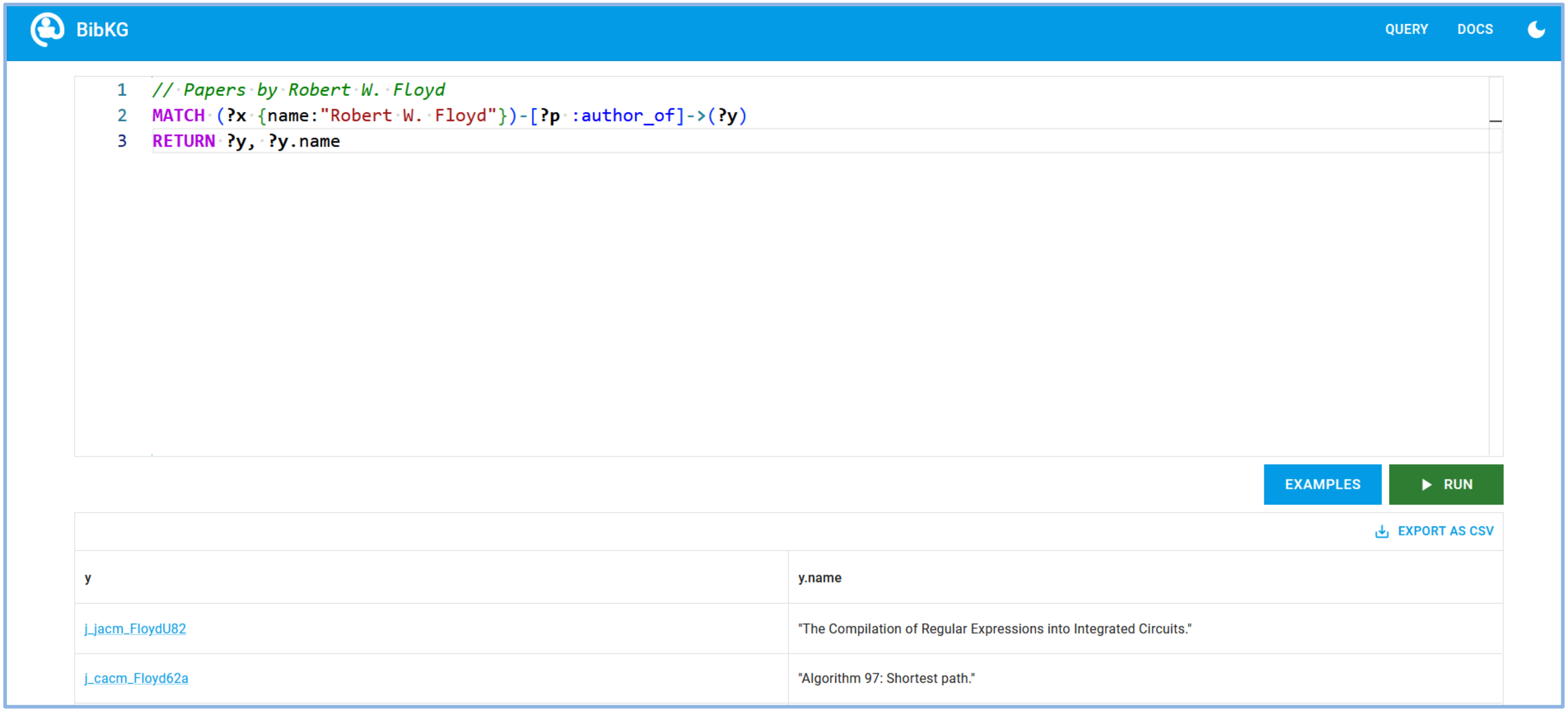


Path queries on property graphs/GQL

GQL:

- Can return paths [GQL, FMRV23]
- Supports powerfull data comparisons over paths [LMV16]
- Many features not well understood yet [GQLDigest]

Let’s see this on BibKG/GQL

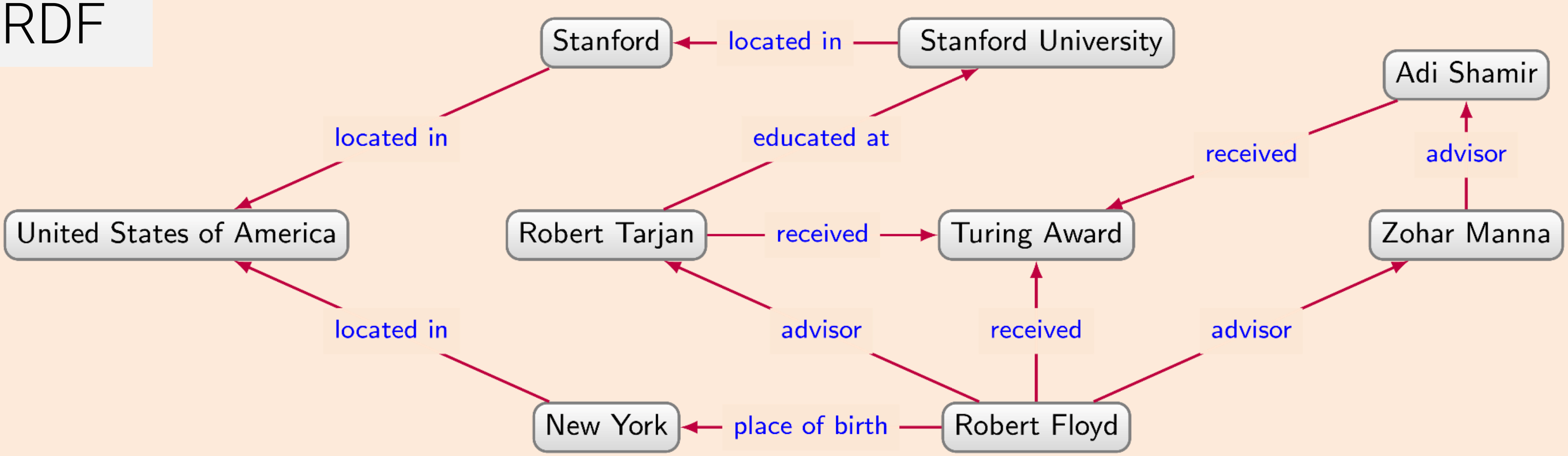


https://bibkg.imfd.cl

Navigational graph patterns

Navigational graph patterns

RDF

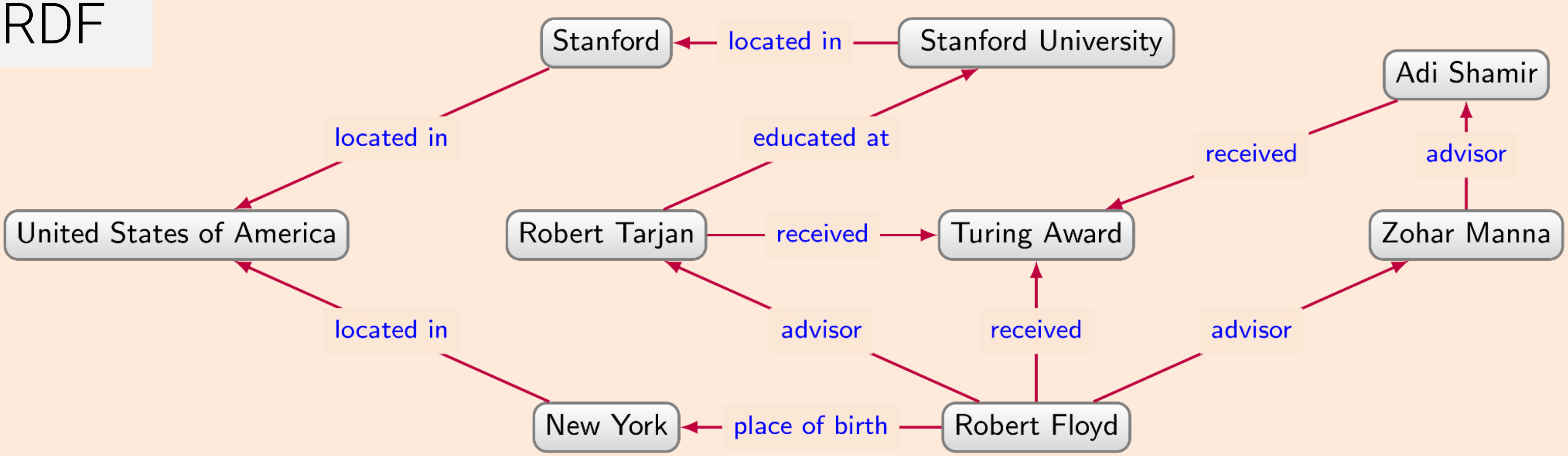


* | / −

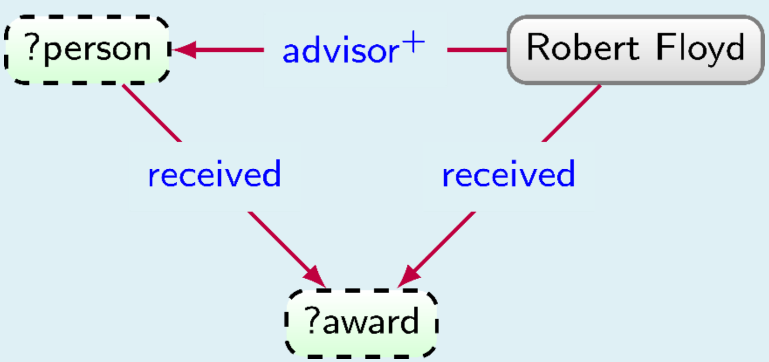
Basic Graph Patterns + Regular Path Queries

Conjunctive regular path queries

RDF

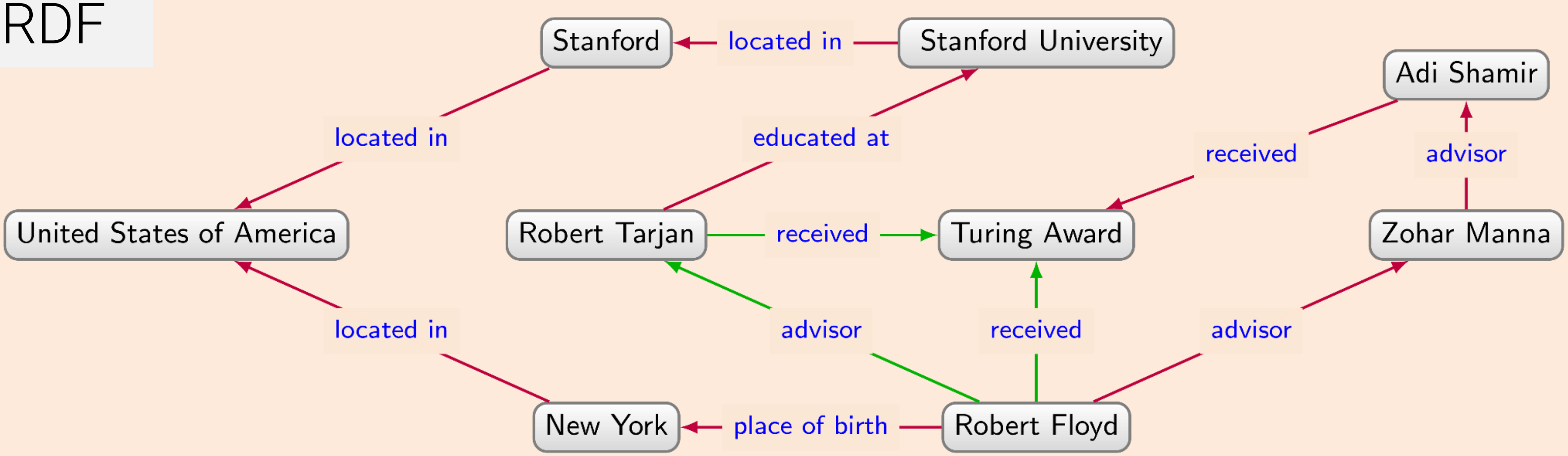


Academic descendants of Robert Floyd who won the same award

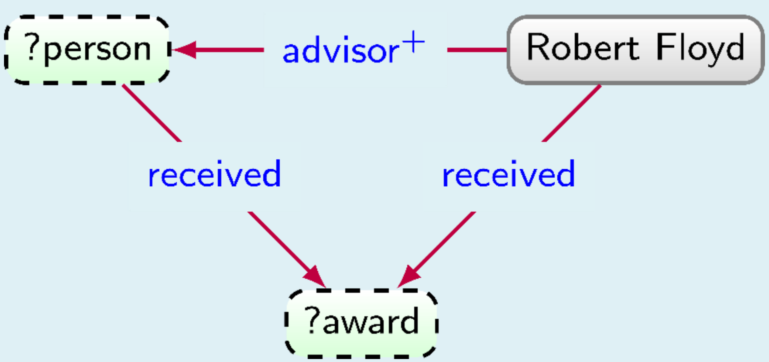


Conjunctive regular path queries

RDF



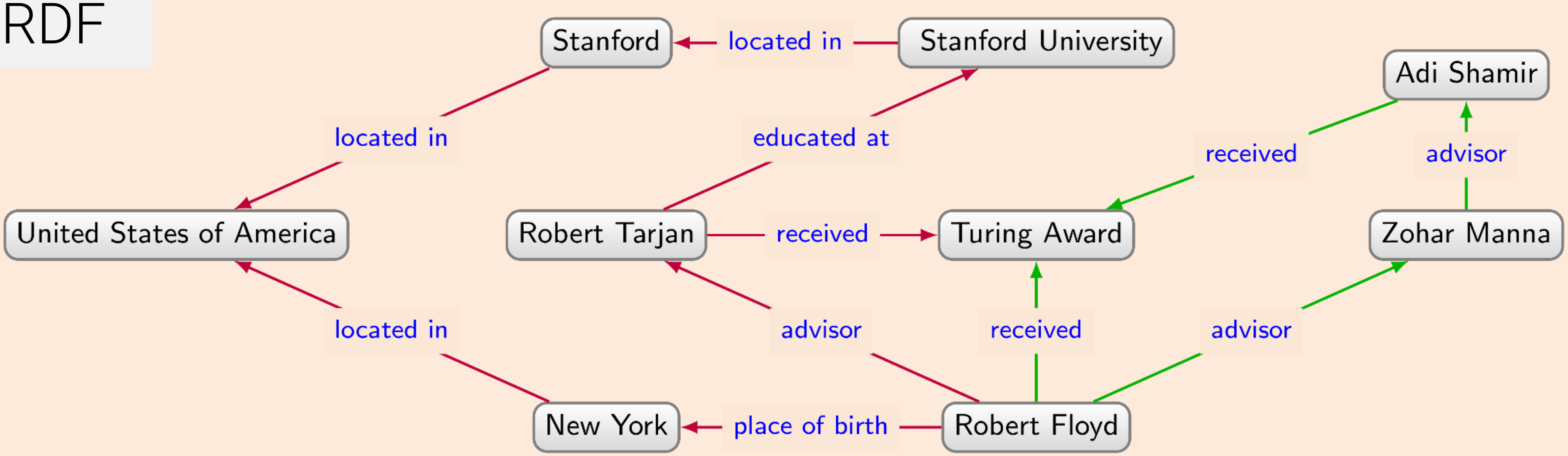
Academic descendants of Robert Floyd who won the same award



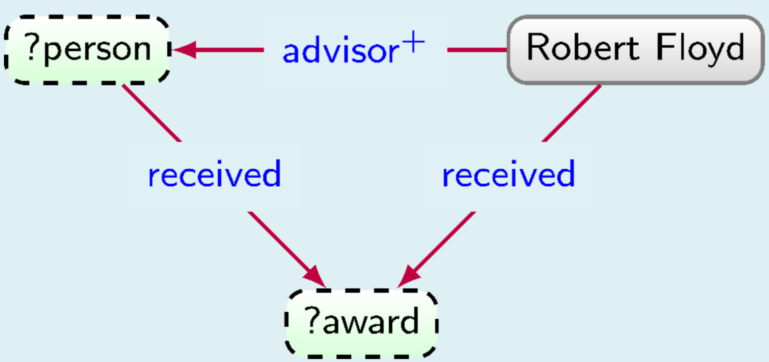
?person	?award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award

Conjunctive regular path queries

RDF



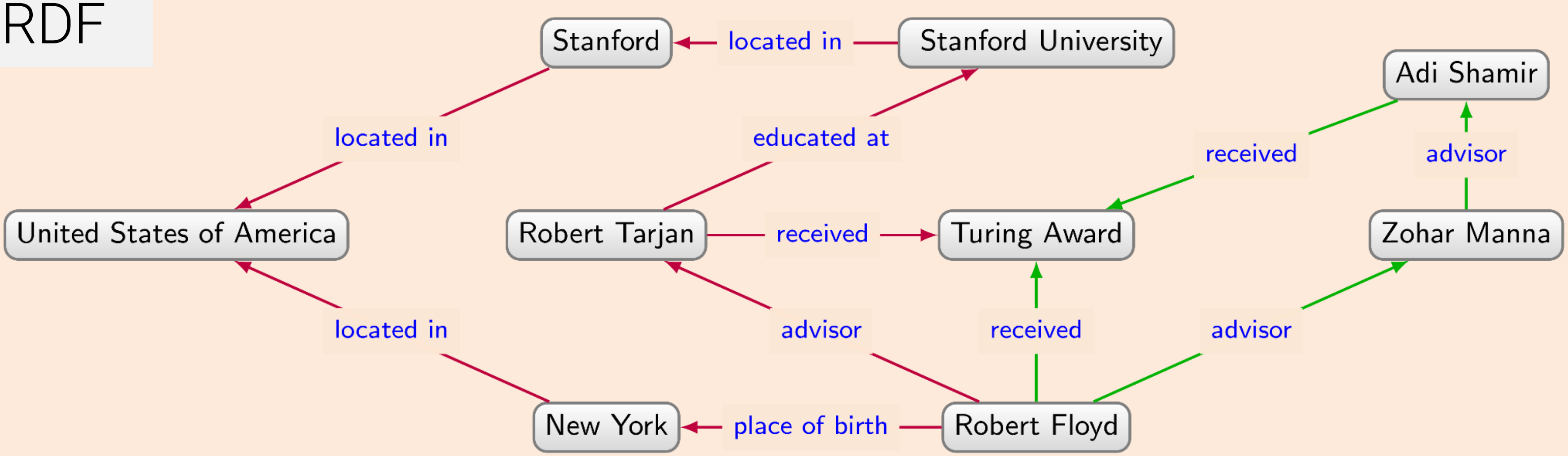
Academic descendants of Robert Floyd who won the same award



?person	?award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award

Conjunctive regular path queries

RDF

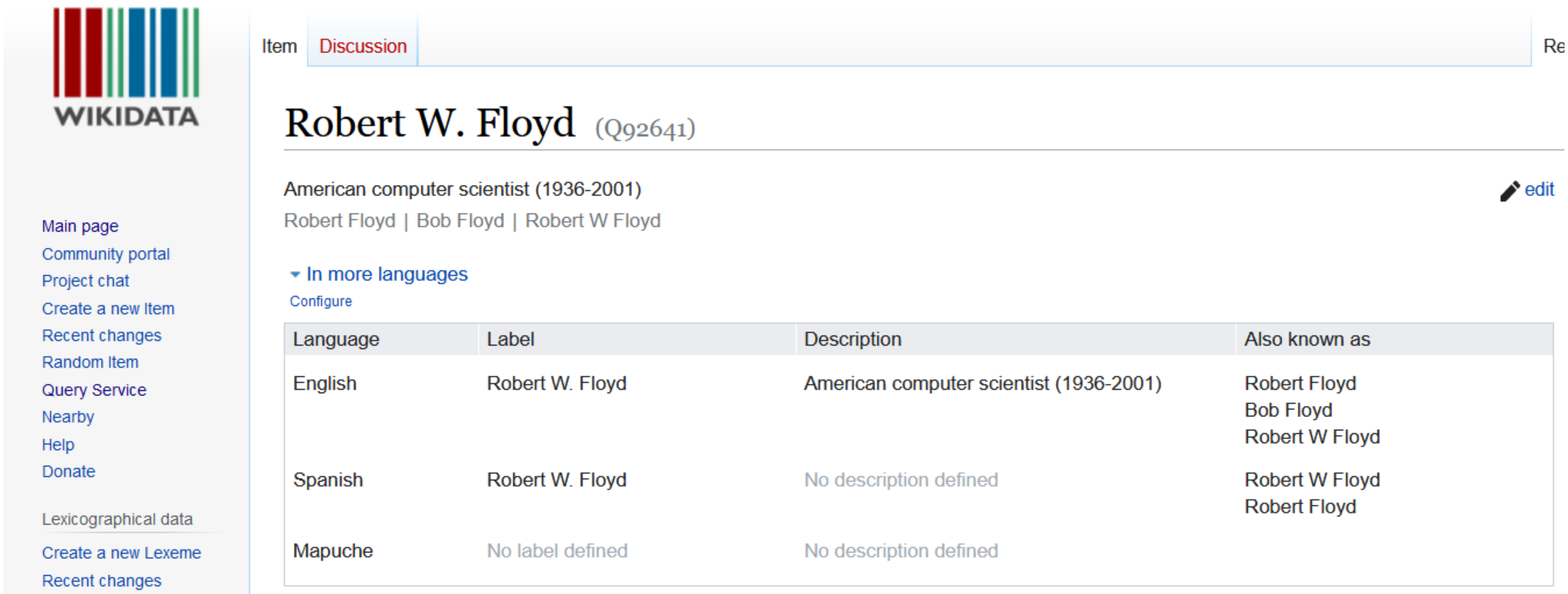


Conjunctive regular path queries (CRPQs)

SPARQL:

- Allows mixing property paths into basic graph patterns
- Known as Conjunctive regular path queries (CRPQs) [CM90]
- Essentially joins with an arbitrary length reachability checks

Let's see this on Wikidata/SPARQL



WIKIDATA

Main page
Community portal
Project chat
Create a new Item
Recent changes
Random Item
Query Service
Nearby
Help
Donate

Lexicographical data

Create a new Lexeme
Recent changes

Item | Discussion | Re

Robert W. Floyd (Q92641)

American computer scientist (1936-2001)

edit

Robert Floyd | Bob Floyd | Robert W Floyd

In more languages

Configure

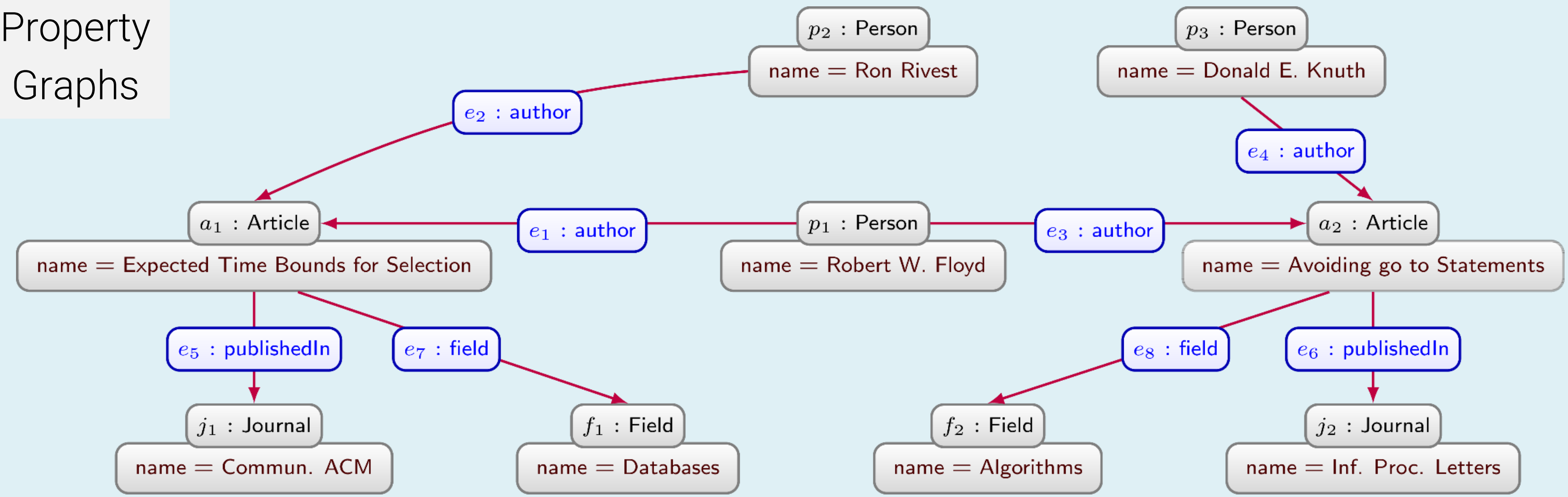
Language	Label	Description	Also known as
English	Robert W. Floyd	American computer scientist (1936-2001)	Robert Floyd Bob Floyd Robert W Floyd
Spanish	Robert W. Floyd	No description defined	Robert W Floyd Robert Floyd
Mapuche	No label defined	No description defined	

https://wikidata.imfd.cl

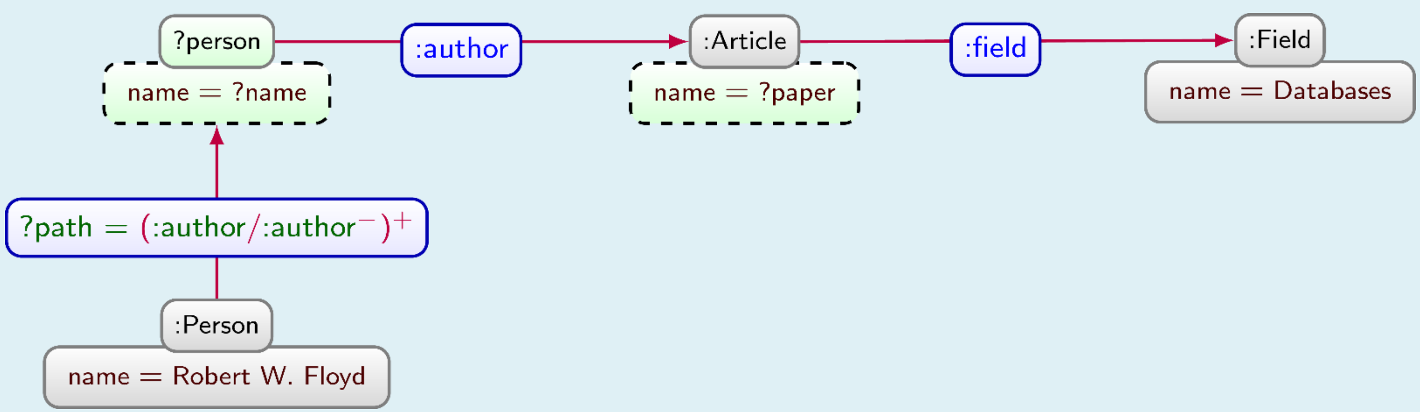
Query1 Query2

CRPQs – but extended

Property Graphs

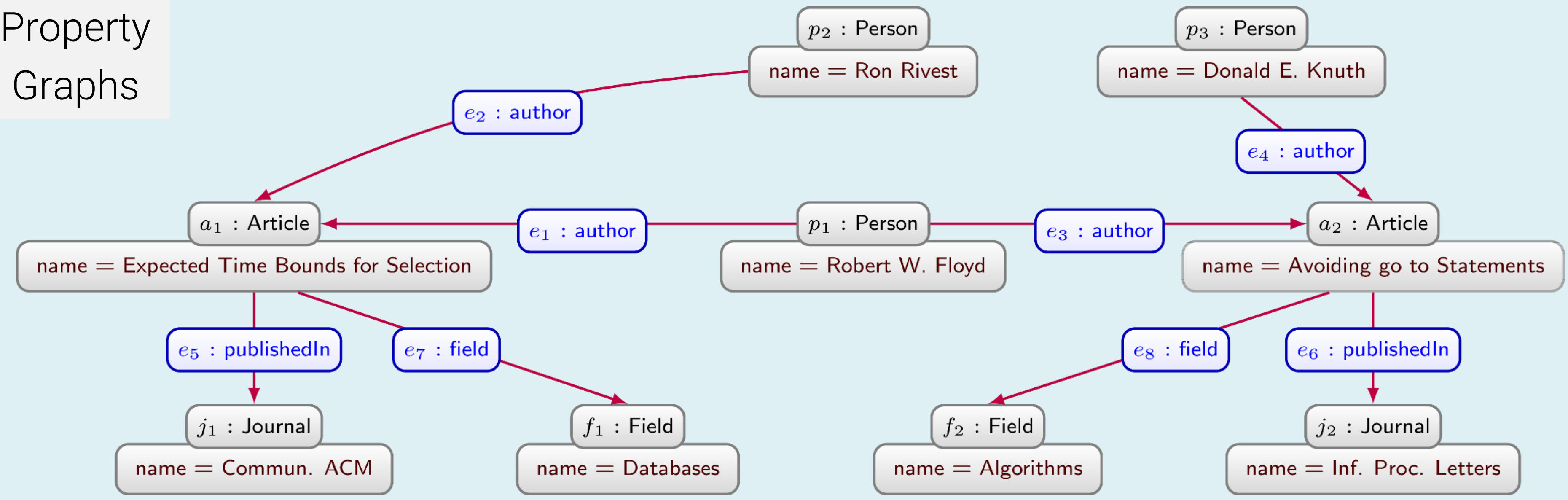


People with a Floyd-number who published a paper about DB

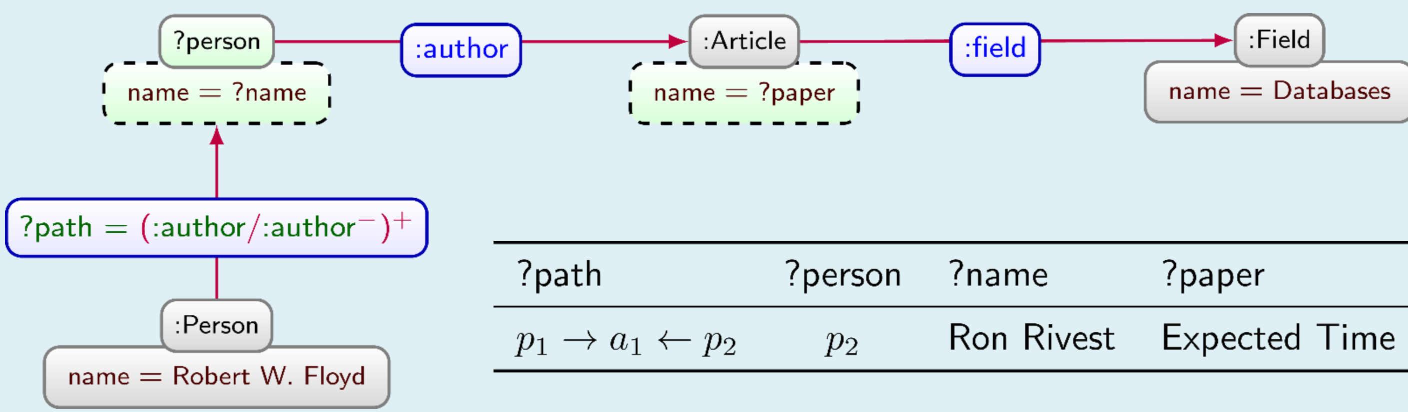


CRPQs – but extended

Property Graphs

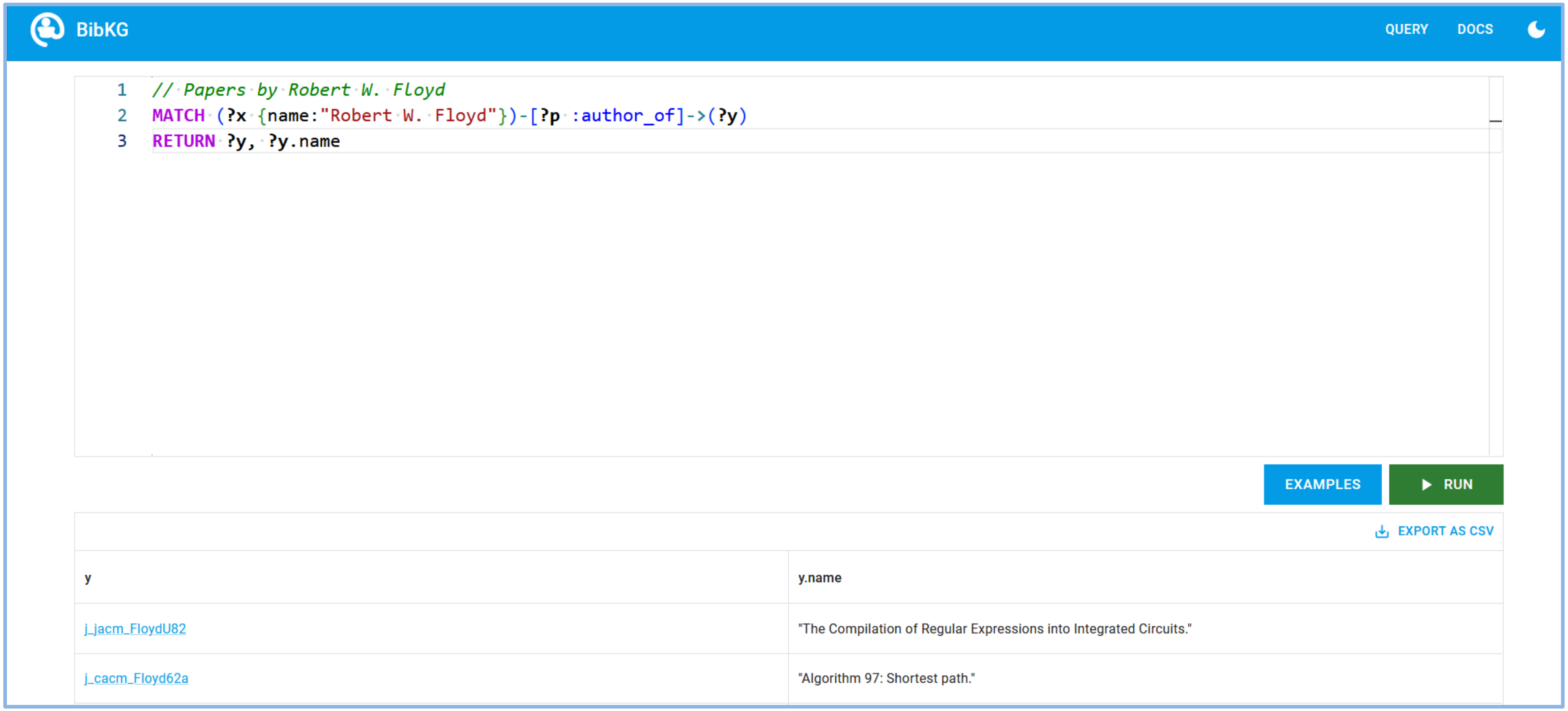


People with a Floyd-number who published a paper about DB



?path	?person	?name	?paper
$p_1 \rightarrow a_1 \leftarrow p_2$	p_2	Ron Rivest	Expected Time Bounds for Selection

Let’s see this on BibKG/GQL



https://bibkg.imfd.cl

Graph Databases:
Complex Graph Patterns

Relational Algebra

At the core of millions of databases we take for granted every day

$\bowtie$ (JOIN)

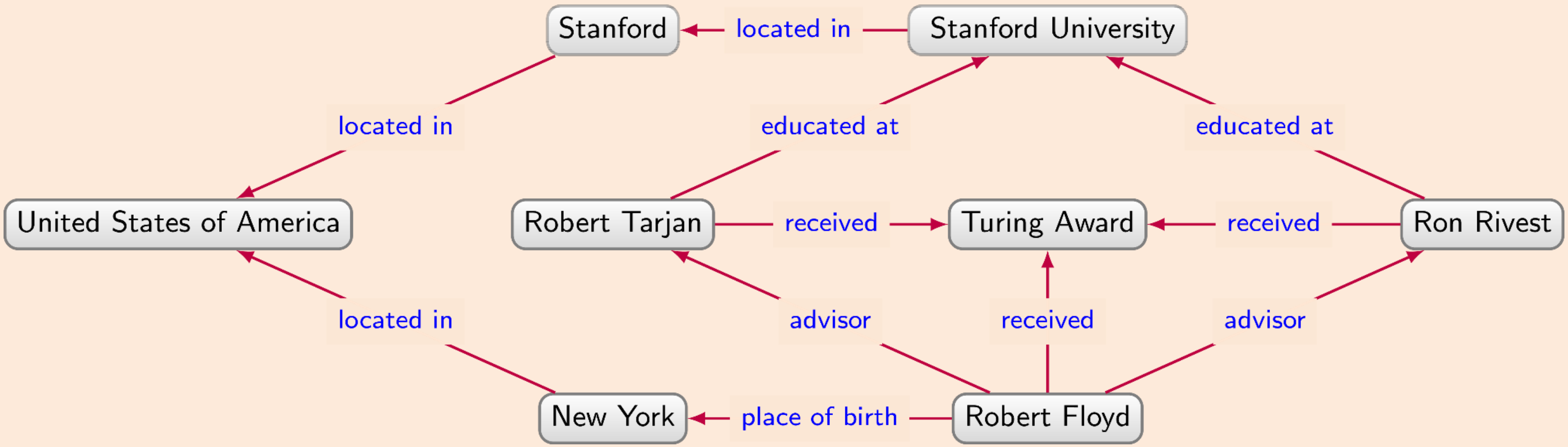
σ (SELECTION)

π (PROJECTION)

$\cup$ (UNION)

$-$ (DIFFERENCE)

Complex graph patterns

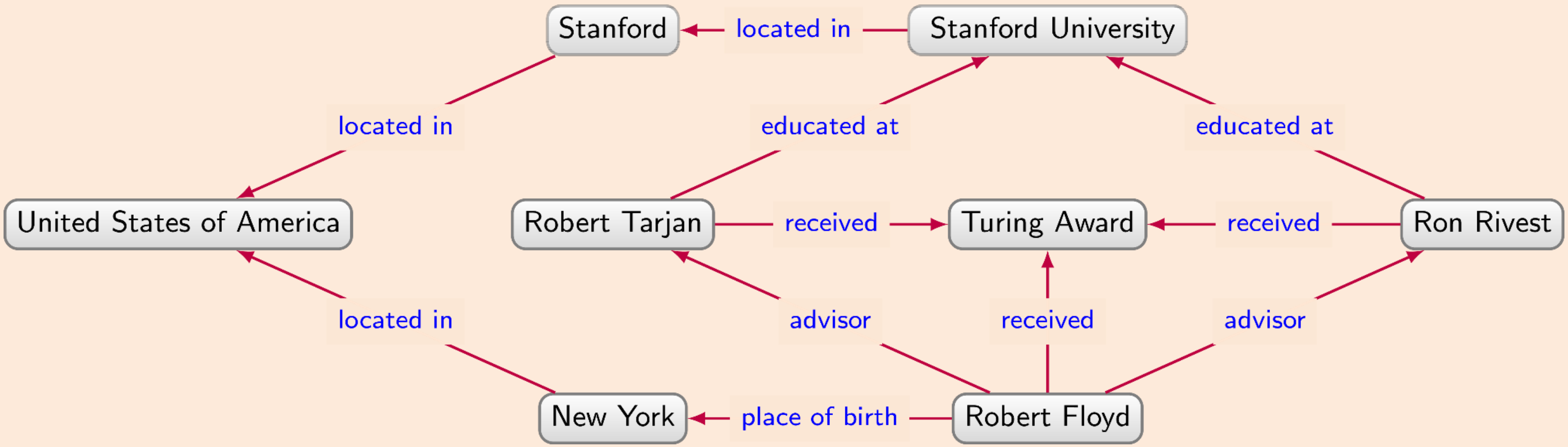


$\bowtie$ $\setminus$ σ π $\cup$

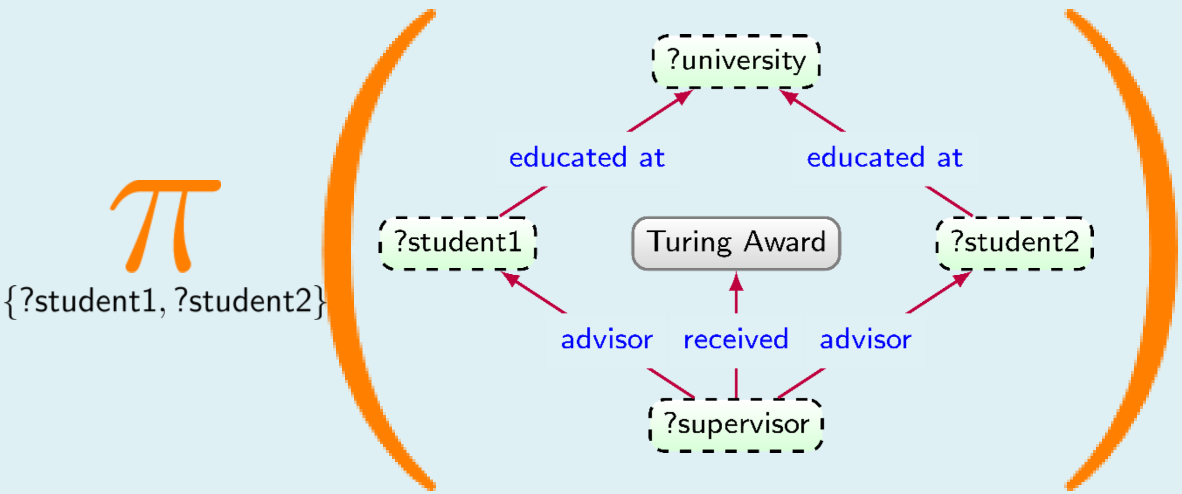
$*$ $|$ $/$ $-$

Graph Patterns + Relational Algebra
+ Regular Path Queries

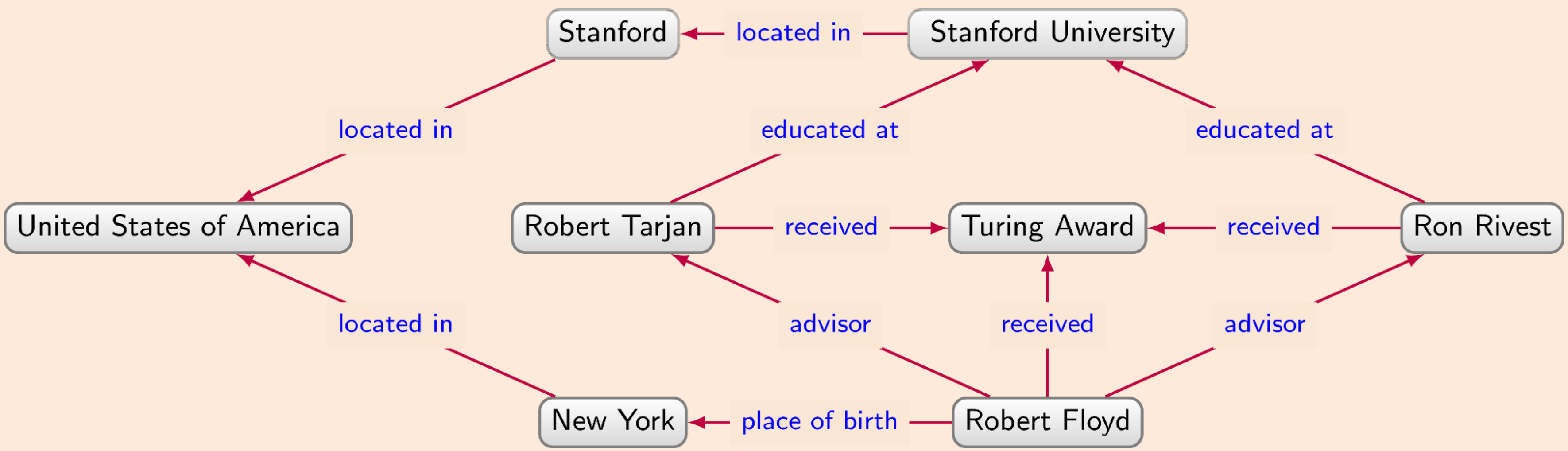
Complex graph patterns



Academic siblings whose supervisor won the Turing Award



Complex graph patterns



Academic siblings whose supervisor won the Turing Award

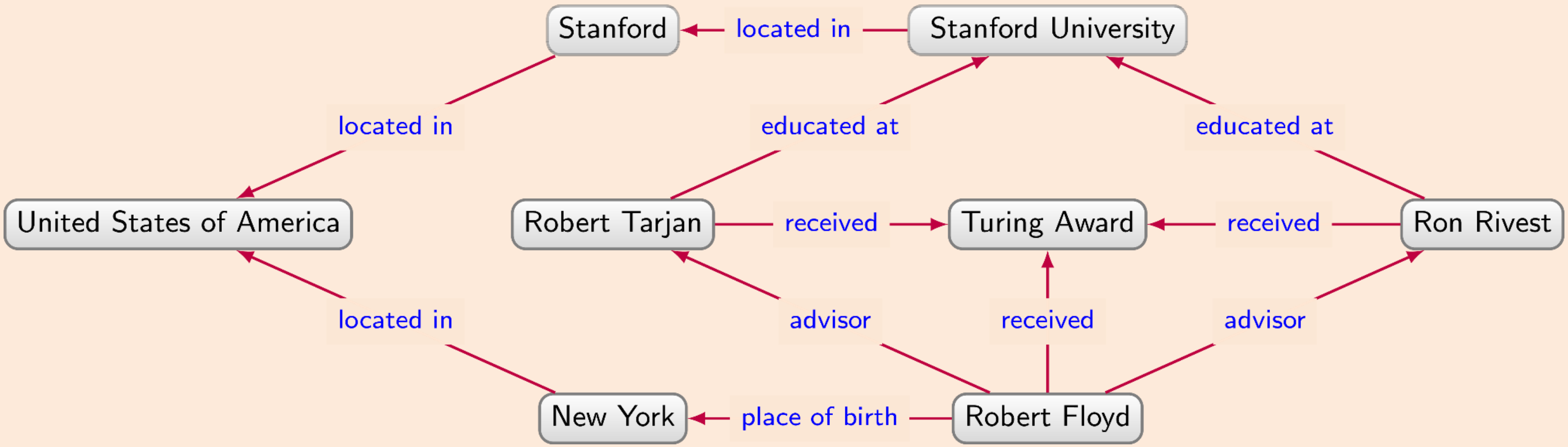
$\pi_{\{?student1,\, ?student2\}}$ (

?supervisor	?student1	?student2	?univeristy
Robert Floyd	Robert Tarjan	Ron Rivest	Stanford Univeritsy
Robert Floyd	Ron Rivest	Robert Tarjan	Stanford Univeritsy
Robert Floyd	Robert Tarjan	Robert Tarjan	Stanford Univeritsy
Robert Floyd	Ron Rivest	Ron Rivest	Stanford Univeritsy

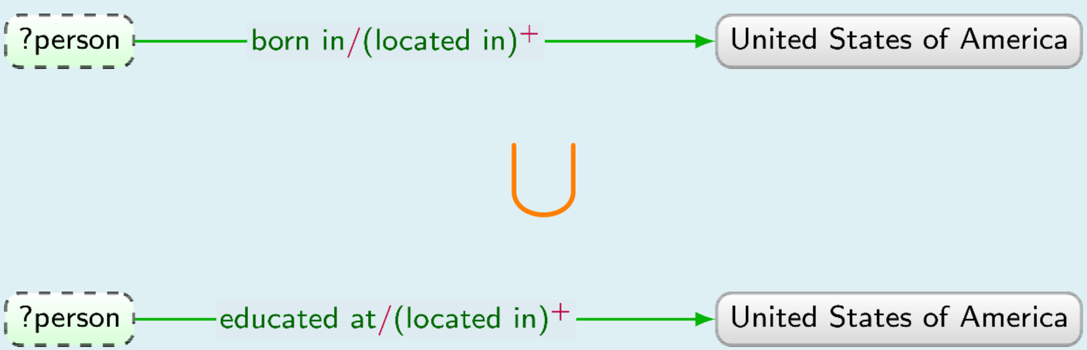
) =

?student1	?student2
Robert Tarjan	Ron Rivest
Ron Rivest	Robert Tarjan
Robert Tarjan	Robert Tarjan
Ron Rivest	Ron Rivest

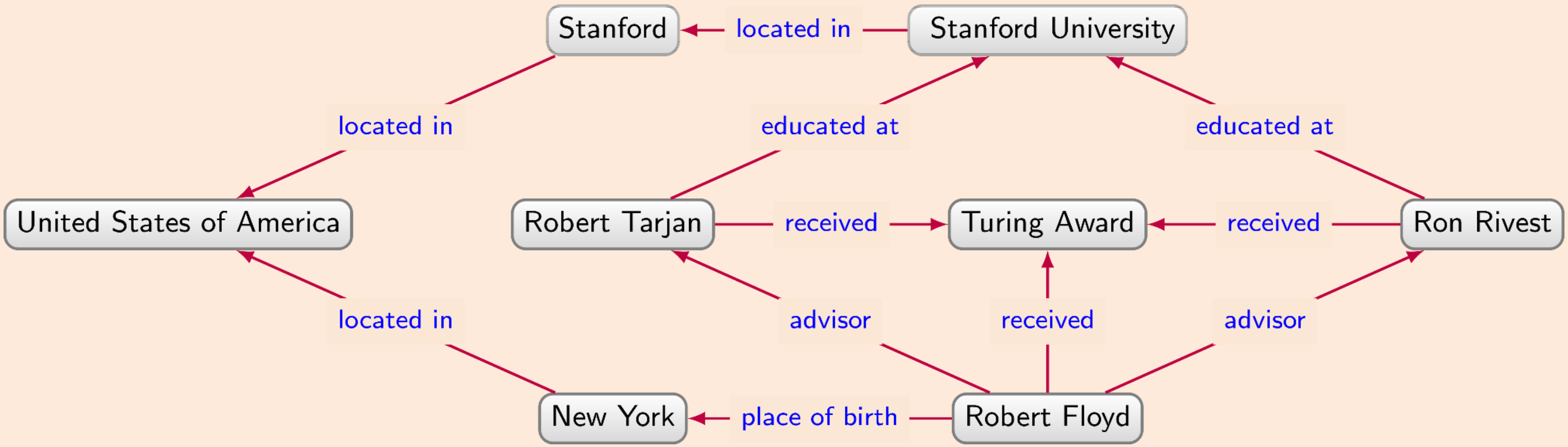
Complex graph patterns



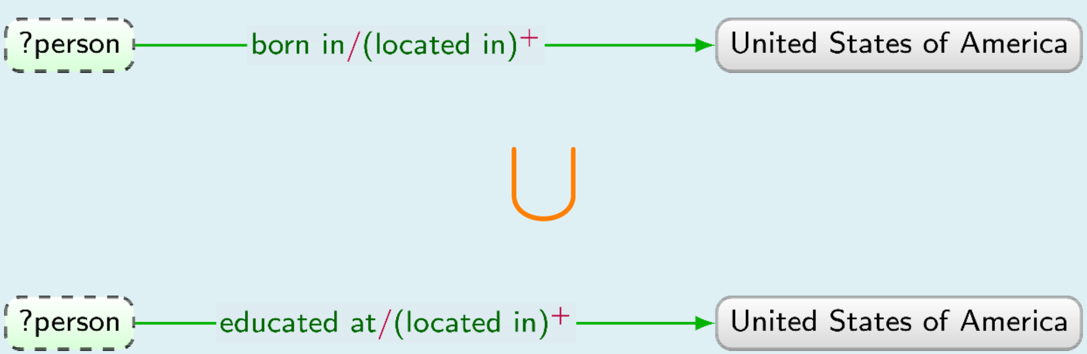
People who were born or studied in the US?



Complex graph patterns



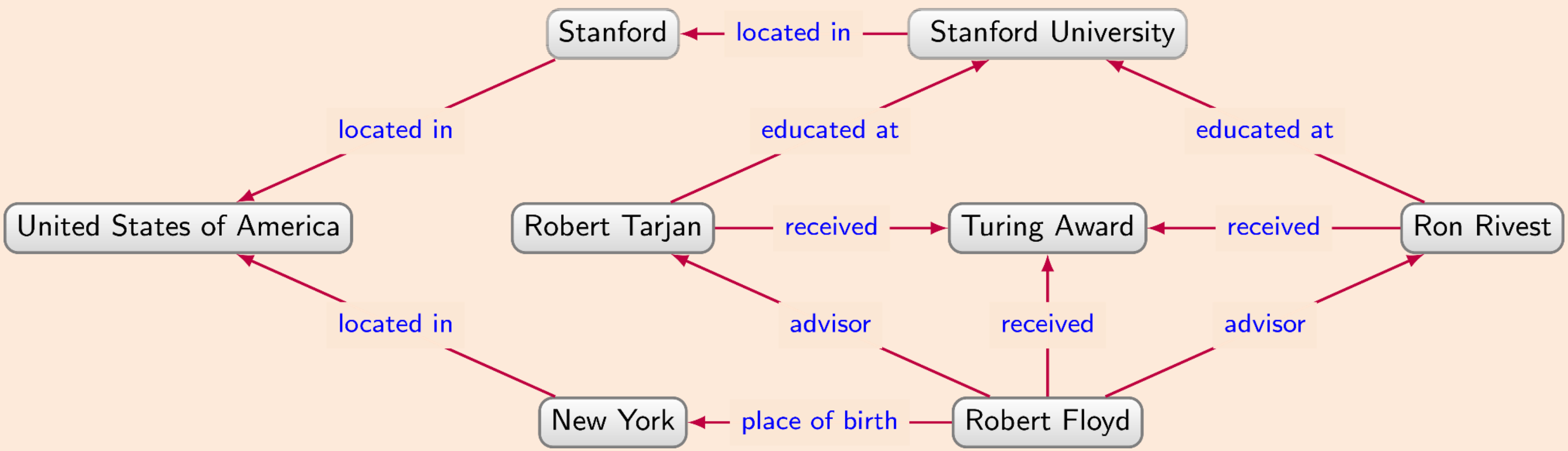
People who were born or studied in the US?



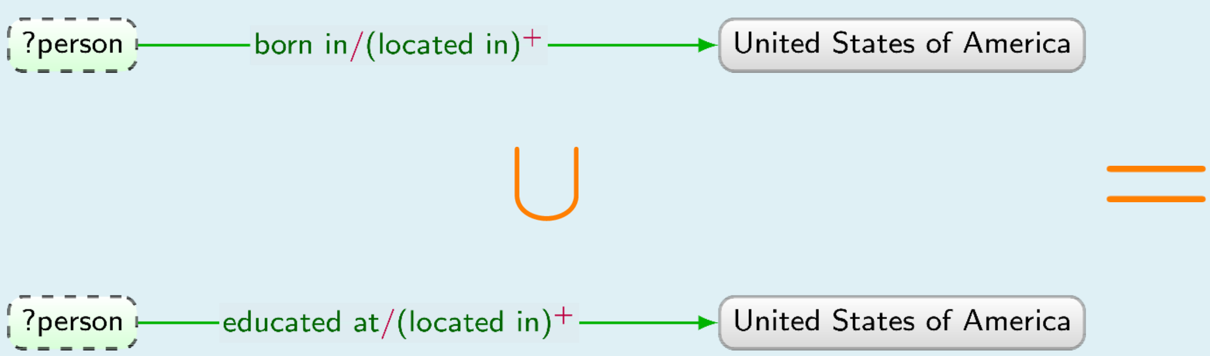
?person
Robert Floyd

?person
Robert Tarjan
Ron Rivest

Complex graph patterns



People who were born or studied in the US?



?person
Robert Floyd
Robert Tarjan
Ron Rivest

Complex graph patterns

- Graph patterns
- Path queries
- Navigational graph patterns
- Relational operations

- **Aggregation**
- ...

Graph languages summary

- RDF/edge-labelled graphs:
  - **SPARQL** W3C standard
  - Bunch of engines (Blazegraph, Jena, Virtuoso, MillenniumDB,...)

- Property graphs:
  - **GQL** ISO standard is still piping hot
  - Very expressive, still being implemented and studied

The floor is yours!

What features are crucial in a graph query language?

Part 1 Conclusions

- Graph databases a hot topic!
- Two models:
  - Directed edge-labelled graphs/RDF
  - Property graphs
- Query features:
  - Basic graph patterns
  - Path queries
  - Relational features

- Need for efficient methods for evaluating queries

Let's learn some efficient methods!

Part 4 spoiler: MillenniumDB

(also, there will be no part 4)

IMFD Chile

- Millennium Science Initiative Chile
  - Interdisciplinary research institue (CS/Social Sciences)
  - Focus on big scale projects
  - One of those: "build a graph database system"

MillenniumDB

- Why us?
  - DB expertise: M. Arenas, J. Reutter, C. Riveros, J. Pérez
  - Semantic Web crowd: A. Hogan, C. Gutierrez, R. Angles
  - Algorithms/compression: G. Navarro, D. Arroyuelo

What for?

- Open source:
  - Build a sandbox for testing research algorithms
  - Test if our research claims check out
  - Support Wikidata
  - Also, this way we can check if theory is worth anything!
- People involved:
  - Carlos Rojas (chief engineer)
  - Vicente Calisto, Gustavo Toro, Benjamín Farías
  - T. Heuer, K. Bosonney, J. Romero, ...
  - Myself (chief complainer)

2019 ...

Key highlights of MillenniumDB

- RDF/SPARQL & Property Graphs/GQL
  - Inside of the same engine
  - SPARQL path queries extended with GQL-inspired features

- Classical database pipeline
  - Quasi-relational

- Focus on support for public query endpoints
  - MVCC-based concurrency control
  - Readers always go through
  - Cental update mechanism

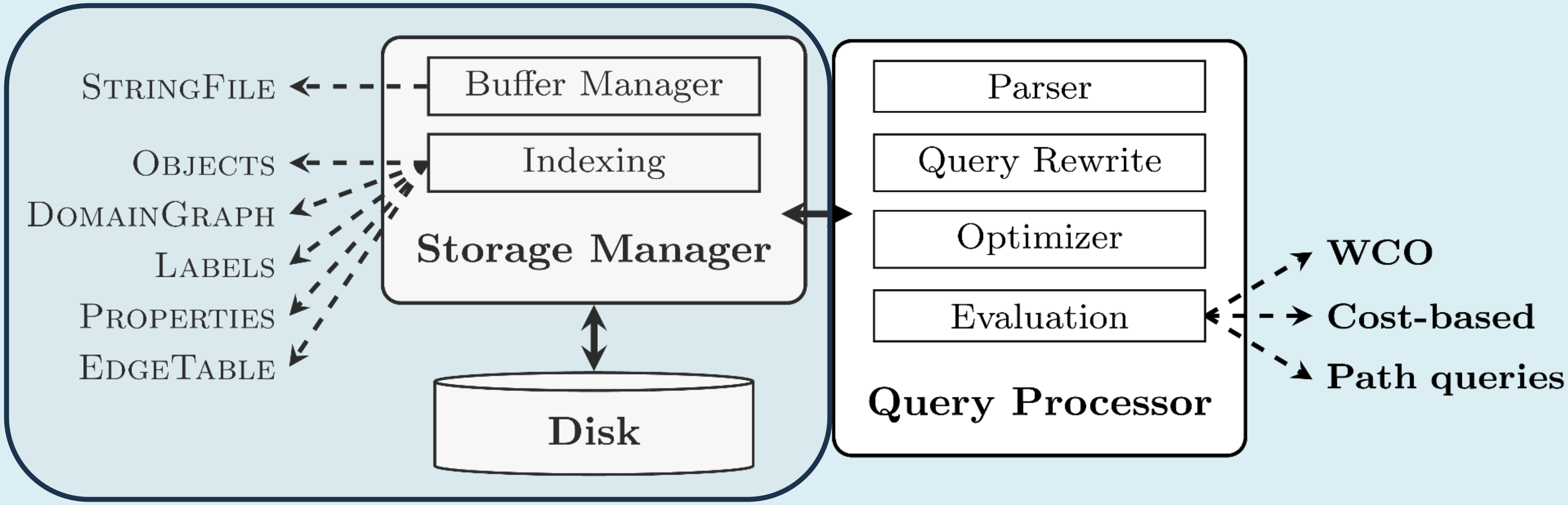
Is theory useful? (no spoiler version)

- Worst-case optimal join processing
  - Graph data usually requires queries where this is useful
  - So will it pan out?
  - Elephant in the room: indices, updates, concurrency

- Path queries
  - An old idea from DB theory that everyone claims they use

- Enumeration algorithms
  - Recent theoretical concept of splitting query evaluation into two
  - Preprocessing with a single pass over the data
  - Enumerate the results one by one (volcano-style)

Architecture of MillenniumDB



RDF **`Triples`**`(subject, predicate, object)`

PGs
`Connections``(src, label, tgt, `<ins>`eId`</ins>`)`
`Labels``(objectId, label)`
`Properties``(`<ins>`objectId`</ins>`, `<ins>`key`</ins>`, value)`

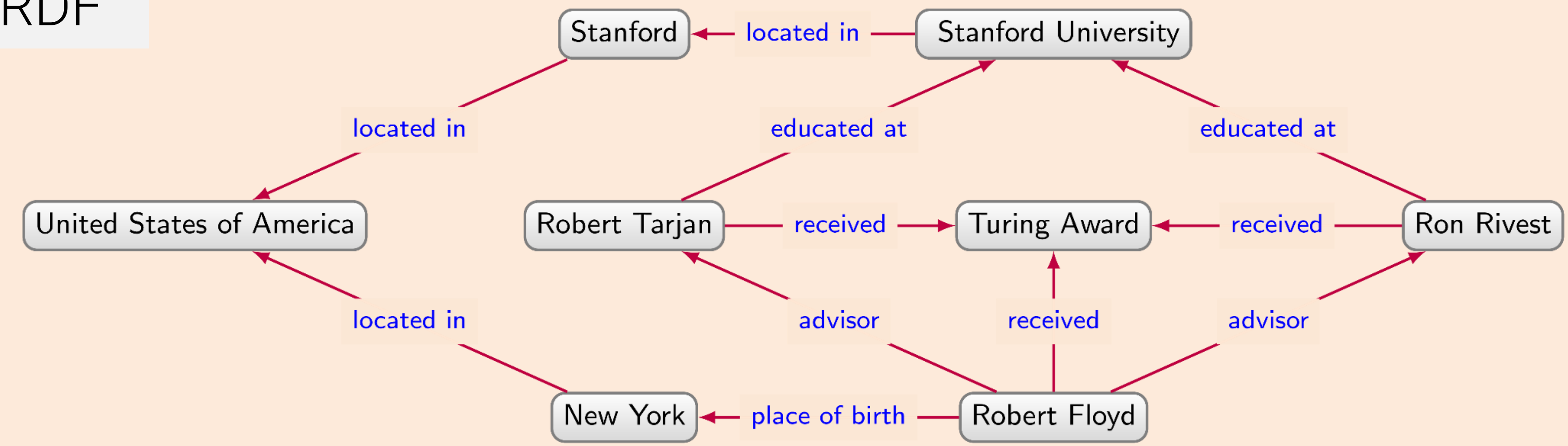
Try it yourself

https://github.com/MillenniumDB/MillenniumDB

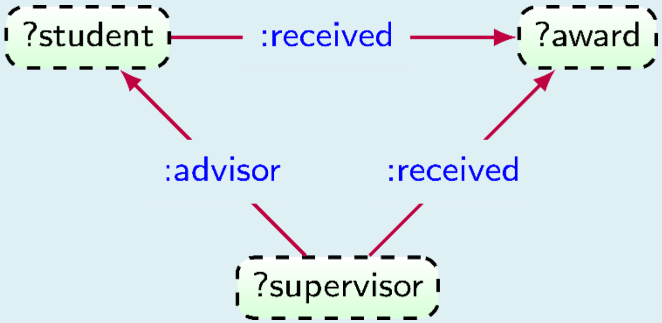
Part 2:
Evaluating Graph Patterns

Evaluating BGPs

RDF



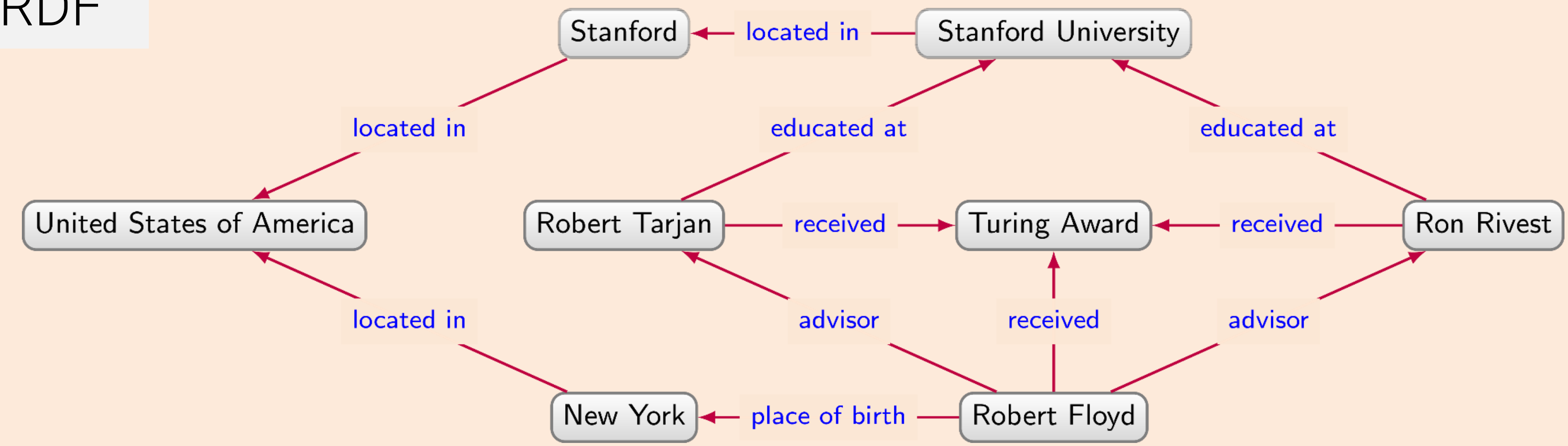
Students and supervisors who both won the same award



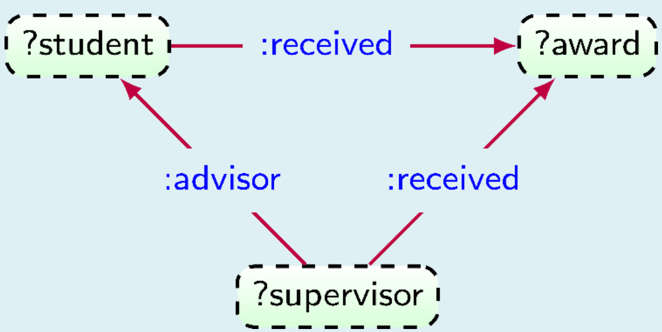
?supervisor	?student	?award
Robert Floyd	Robert Tarjan	Turing Award
Robert Floyd	Ron Rivest	Turing Award

Evaluating BGPs

RDF



Students and supervisors who both won the same award




```
SELECT *
WHERE {
  ?supervisor :advisor  ?student .
  ?supervisor :received ?award .
  ?student    :received ?award .
}
```

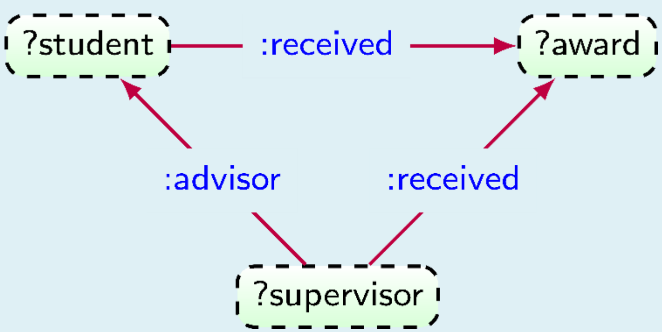
How is this stored?

RDF

`Triples``(subject, predicate, object)`

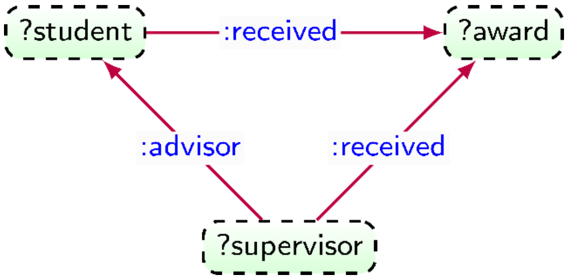
- Graph stored as a relation
- Graph pattern is a join of this relation
- And usually we do this join many times

Students and supervisors who both won the same award




```
SELECT *
WHERE {
  ?supervisor :advisor  ?student .
  ?supervisor :received ?award .
  ?student    :received ?award .
}
```

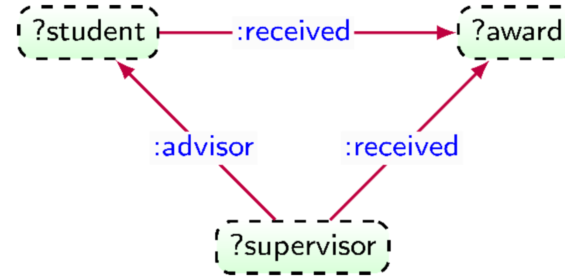
Graphs as relations



Triples

subject	predicate	object
Robert Floyd	advisor	Robert Tarjan
Robert Floyd	advisor	Adi Shamir
John Hopcroft	advisor	Alfred Aho
Robert Floyd	received	Turing Award
Robert Tarjan	received	Turing Award
Adi Shamir	received	Turing Award
John Hopcroft	received	Turing Award
Alfred Aho	received	Turing Award

Graphs as relations



phds

s	o
Robert Floyd	Robert Tarjan
Robert Floyd	Adi Shamir
John Hopcroft	Alfred Aho

$\pi_{s,o}(\sigma_{p=\text{advisor}}(\text{Triples}))$

won1

s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

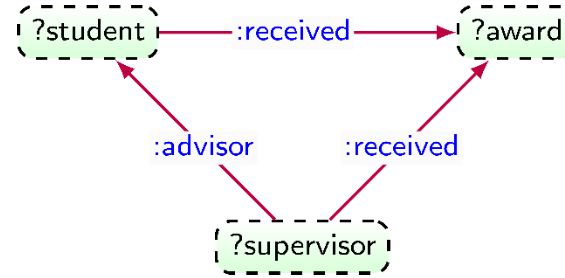
$\pi_{s,o}(\sigma_{p=\text{received}}(\text{Triples}))$

won2

s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

$\pi_{s,o}(\sigma_{p=\text{received}}(\text{Triples}))$

Graphs as relations



phds

s	o
Robert Floyd	Robert Tarjan
Robert Floyd	Adi Shamir
John Hopcroft	Alfred Aho

$\pi_{s,o}(\sigma_{p=\text{advisor}}(\text{Triples}))$

won1

s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

$\pi_{s,o}(\sigma_{p=\text{received}}(\text{Triples}))$

won2

s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

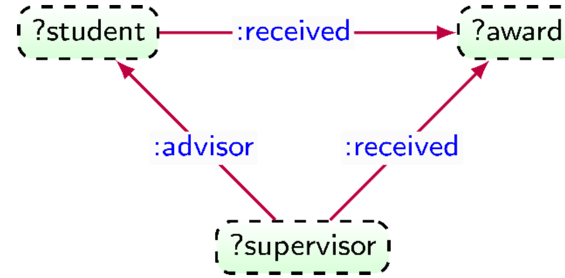
$\pi_{s,o}(\sigma_{p=\text{received}}(\text{Triples}))$

$$\textbf{phds} \bowtie_{\textbf{phds}.\text{s} = \textbf{won1}.\text{s}} \textbf{won1}$$

phdWon

phds.s	phds.o	won1.s	won1.o
Robert Floyd	Robert Tarjan	Robert Floyd	Turing Award
Robert Floyd	Adi Shamir	Robert Floyd	Turing Award
John Hopcroft	Alfred Aho	John Hopcroft	Turing Award

Graphs as relations



$$\textbf{phds} \bowtie_{\textbf{phds}.\mathsf{s} = \textbf{won1}.\mathsf{s}} \textbf{won1}$$

phdWon

phds.s	phds.o	won1.s	won1.o
Robert Floyd	Robert Tarjan	Robert Floyd	Turing Award
Robert Floyd	Adi Shamir	Robert Floyd	Turing Award
John Hopcroft	Alfred Aho	John Hopcroft	Turing Award

won2

s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

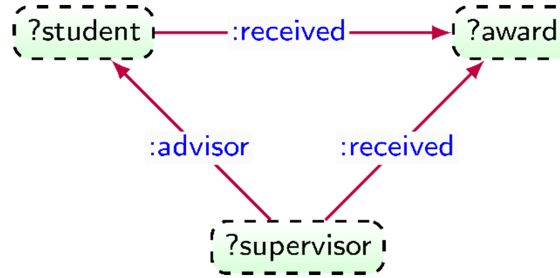
$\pi_{s,o}(\sigma_{p=\mathsf{received}}(\mathsf{Triples}))$

$$\textbf{phdWon} \bowtie_{\textbf{phds}.\mathsf{o} = \textbf{won2}.\mathsf{s} \wedge \textbf{won1}.\mathsf{o} = \textbf{won2}.\mathsf{o}} \textbf{won2}$$

allTheData

phds.s	phds.o	won1.s	won1.o	won2.s	won2.o
Robert Floyd	Robert Tarjan	Robert Floyd	Turing Award	Robert Tarjan	Turing Award
Robert Floyd	Adi Shamir	Robert Floyd	Turing Award	Adi Shamir	Turing Award
John Hopcroft	Alfred Aho	John Hopcroft	Turing Award	Alfred Aho	Turing Award

Graphs as relations



$$\textbf{phds} \bowtie_{\textbf{phds}.s = \textbf{won1}.s} \textbf{won1}$$

phdWon

phds.s	phds.o	won1.s	won1.o
Robert Floyd	Robert Tarjan	Robert Floyd	Turing Award
Robert Floyd	Adi Shamir	Robert Floyd	Turing Award
John Hopcroft	Alfred Aho	John Hopcroft	Turing Award

won2

s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

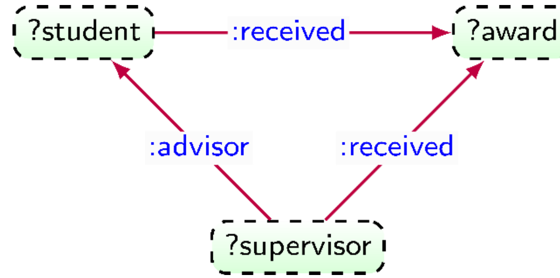
$\pi_{s,o}(\sigma_{p=\text{received}}(\text{Triples}))$

$$\textbf{phdWon} \bowtie_{\textbf{phds}.o = \textbf{won2}.s \wedge \textbf{won1}.o = \textbf{won2}.o} \textbf{won2}$$

whatWeWant

supervisor	student	commonAward
Robert Floyd	Robert Tarjan	Turing Award
Robert Floyd	Adi Shamir	Turing Award
John Hopcroft	Alfred Aho	Turing Award

Notation for join queries



advisor

s	o
Robert Floyd	Robert Tarjan
Robert Floyd	Adi Shamir
John Hopcroft	Alfred Aho

$\pi_{s,o}(\sigma_{p=\text{advisor}}(\text{Triples}))$

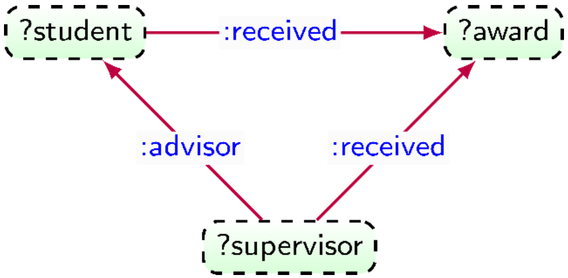
received

s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

$\pi_{s,o}(\sigma_{p=\text{received}}(\text{Triples}))$

advisor(?x,?y), **received**(?x,?z), **received**(?y, ?z)

Notation for join queries



advisor

s	o
Robert Floyd	Robert Tarjan
Robert Floyd	Adi Shamir
John Hopcroft	Alfred Aho

$\pi_{s,o}(\sigma_{p=\text{advisor}}(\text{Triples}))$

received

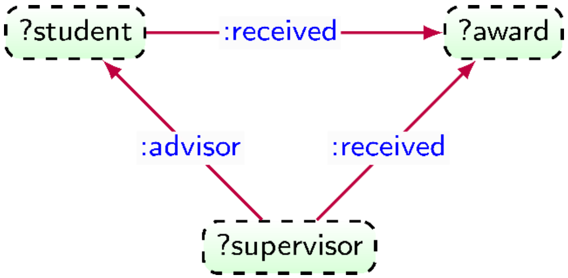
s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

$\pi_{s,o}(\sigma_{p=\text{received}}(\text{Triples}))$

$$\textbf{advisor}(?x,?y),\ \textbf{received}(?x,?z),\ \textbf{received}(?y,\ ?z)$$

$$\textbf{advisor}(?x,?y) \bowtie \textbf{received}(?x,?z) \bowtie \textbf{received}(?y,\ ?z)$$

Notation for join queries



advisor

s	o
Robert Floyd	Robert Tarjan
Robert Floyd	Adi Shamir
John Hopcroft	Alfred Aho

$\pi_{s,o}(\sigma_{p=\text{advisor}}(\text{Triples}))$

received

s	o
Robert Floyd	Turing Award
Robert Tarjan	Turing Award
Adi Shamir	Turing Award
John Hopcroft	Turing Award
Alfred Aho	Turing Award

$\pi_{s,o}(\sigma_{p=\text{received}}(\text{Triples}))$

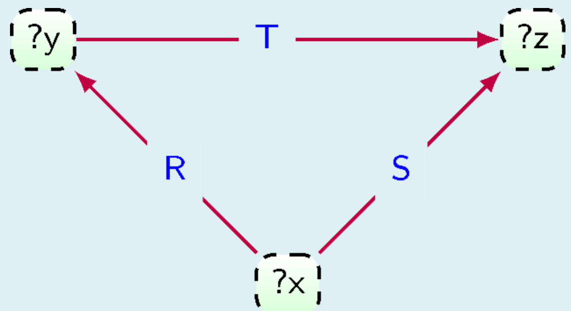
advisor(?supervisor,?student), **received**(?supervisor,?award), **received**(?student, ?award)

whatWeWant

?supervisor	?student	?award
Robert Floyd	Robert Tarjan	Turing Award
Robert Floyd	Adi Shamir	Turing Award
John Hopcroft	Alfred Aho	Turing Award

Notation for join queries

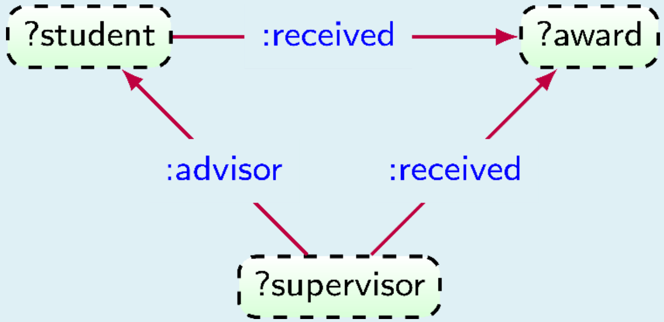
- Basically, joins are important
- Graph patterns can be viewed as joins of binary relations



$$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$$

$$\mathbf{R}(?x,?y),\ \mathbf{S}(?x,?z),\ \mathbf{T}(?y,\ ?z)$$

How many results can a join query have?

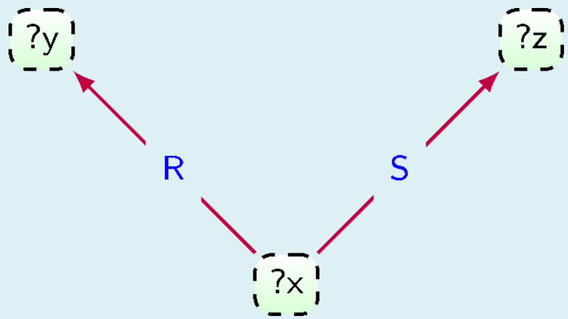


advisor(?supervisor,?student) $\bowtie$ **received**(?supervisor,?award)

Over graphs with a fixed budget $n = 4$ for each edge

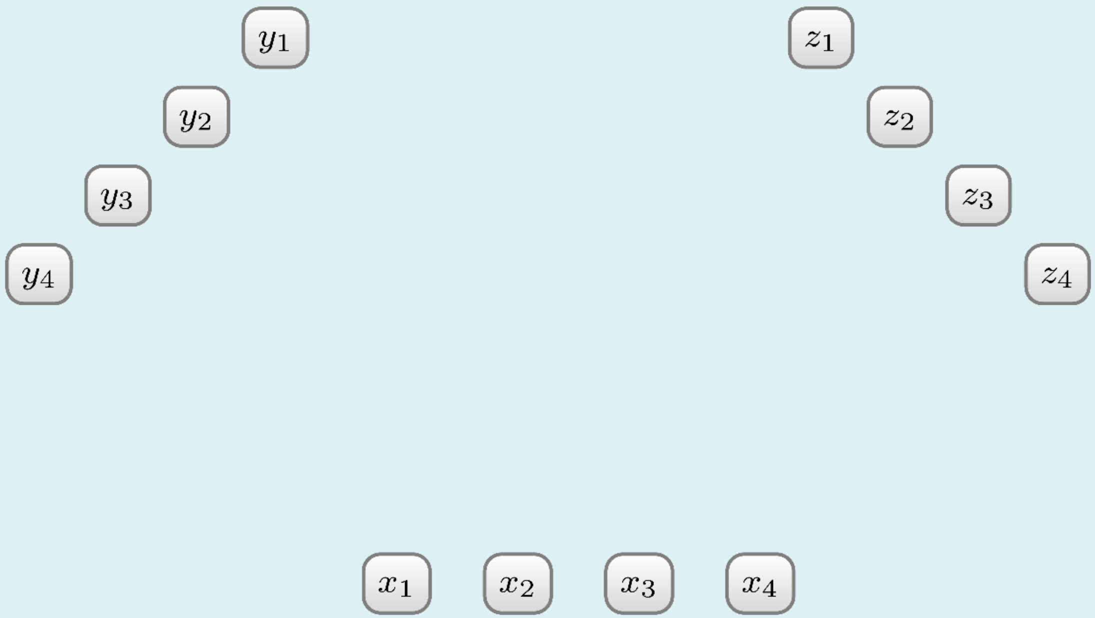
- This just means $|\textbf{advisor}| = |\textbf{received}| = 4$
- Turns out this is a very subtle question!

How many results can a join query have?

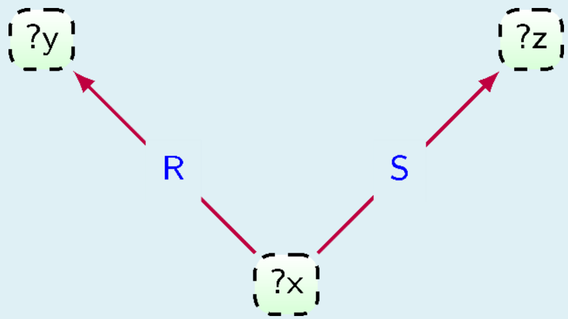


$$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z)$$

Over graphs with a fixed budget $n = 4$ for each edge

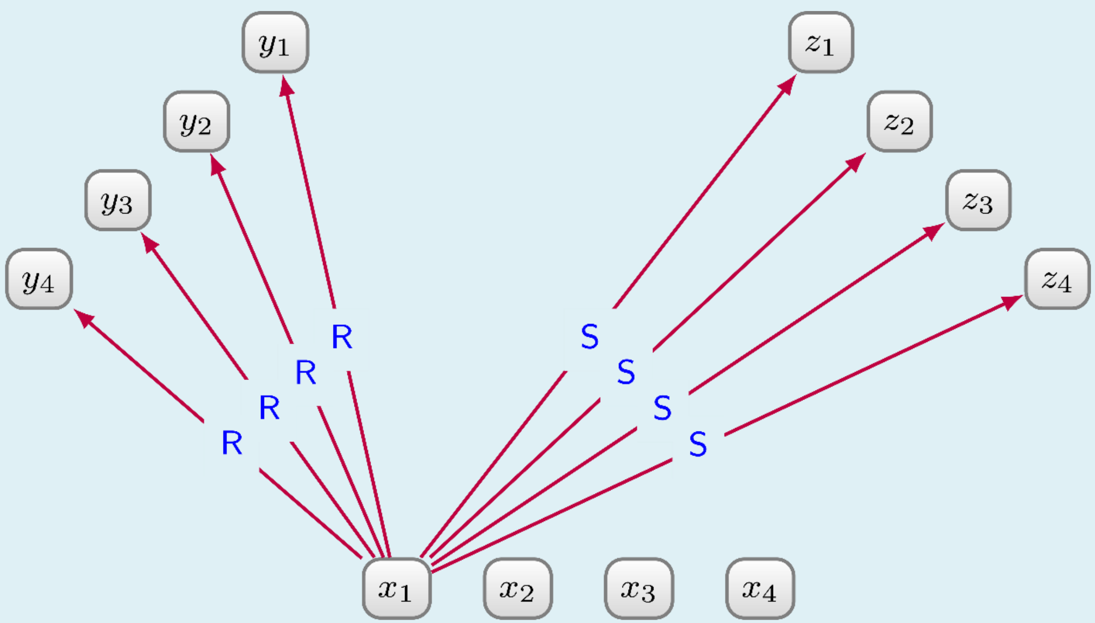


How many results can a join query have?

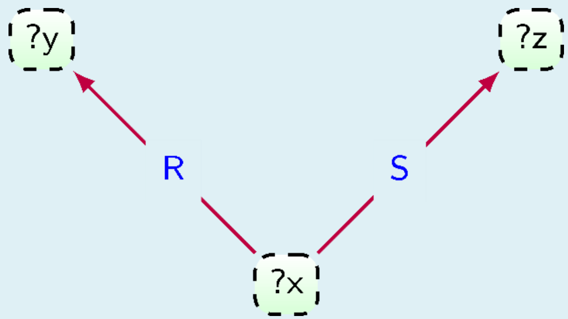


$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z)$

Over graphs with a fixed budget $n = 4$ for each edge



How many results can a join query have?



$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z)$

Over graphs with a fixed budget $n = 4$ for each edge

R

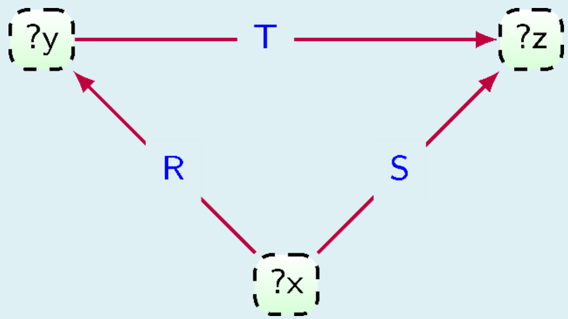
?x	?z
x_1	y_1
x_1	y_2
x_1	y_3
x_1	y_4

$$\bowtie$$

S

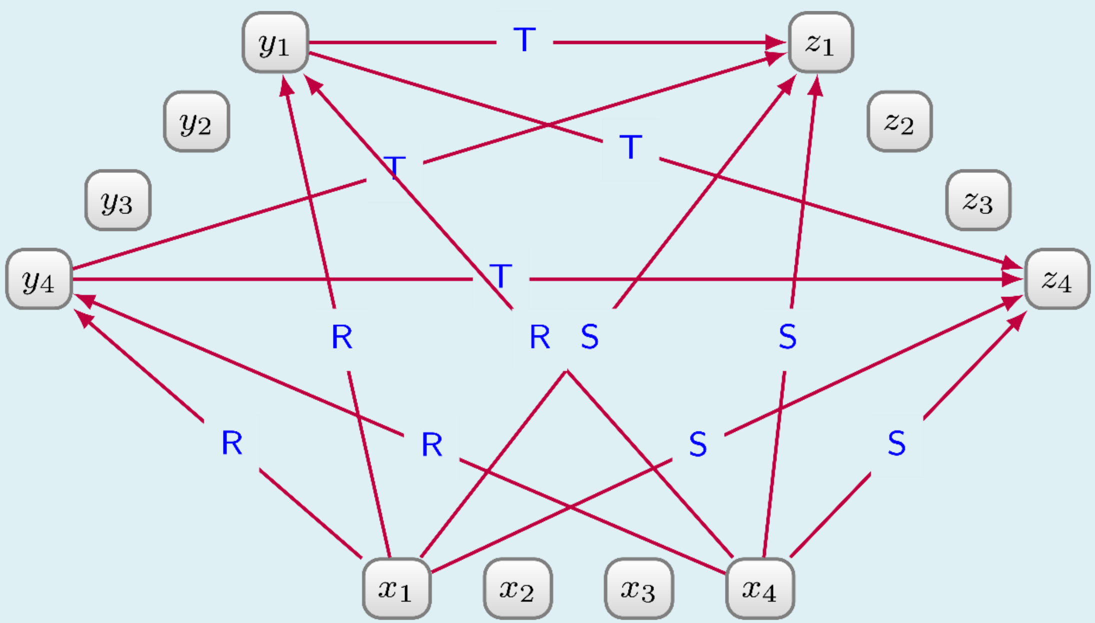
?x	?z
x_1	z_1
x_1	z_2
x_1	z_3
x_1	z_4

And now?

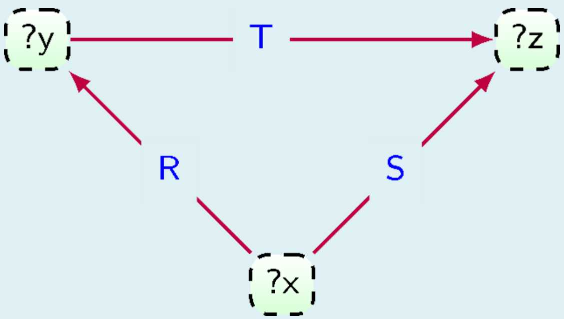


$$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$$

Over graphs with a fixed budget $n = 4$ for each edge



And now?



$$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$$

Over graphs with a fixed budget $n = 4$ for each edge

R

?x	?y
x_1	y_1
x_1	y_4
x_4	y_1
x_4	y_4

$\bowtie$

S

?x	?z
x_1	z_1
x_1	z_4
x_4	z_1
x_4	z_4

$\bowtie$

T

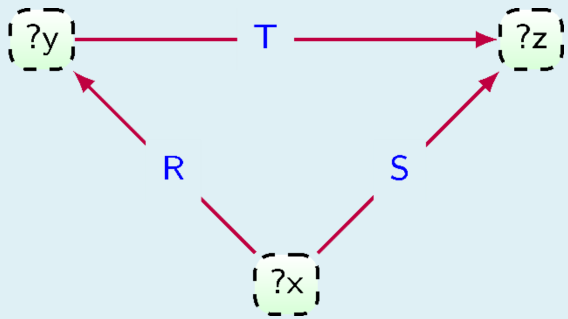
?y	?z
y_1	z_1
y_1	z_4
y_4	z_1
y_4	z_4

$=$

output

?x	?y	?z
x_1	y_1	z_1
x_1	y_1	z_4
x_1	y_4	z_1
x_1	y_4	z_4
x_4	y_1	z_1
x_4	y_1	z_4
x_4	y_4	z_1
x_4	y_4	z_4

And now?



$$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$$

Over graphs with a fixed budget $n = 4$ for each edge

- In this instance we got 8!
- Interestingly, this is the maximum.

Why?

AGM bound

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

assume $|\mathbf{R}_i| = n_i$, where $n_1, \ldots, n_k$ are fixed

$w_1^m, \ldots, w_k^m$ is a **solution for the LP**:

$$\begin{array}{lll} \text{minimize:} & n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k} & \\ \text{such that:} & \displaystyle\sum_{R_i : y \in \overline{x}_i} w_i \geq 1 & \text{(for every variable } y\text{)} \\ & 0 \leq w_i \leq 1 & (i = 1, \ldots, k) \end{array}$$

- $|Q(D)| \leq |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (for all such D)
- $|Q(D)| = |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (on one such D)

???

See [AGM08] for details

What would be ideal?

- Best possible algorithm for a query Q :
  - $O(1)$ per query result
  - So runtime would be $O(|Q(D)|)$ on any instance D
  - This is the holy grail of databases!
    - So it probably does not exist

But let us try to see how good this would be

(i.e. let’s see how many results there are)

Estimating the output size

$$Q = \mathbf{R}_1(\text{?x,?y}) \bowtie \mathbf{R}_2(\text{?y,?z})$$

$$|\mathbf{R}_1| = |\mathbf{R}_2| = n \qquad \Rightarrow \qquad |Q(D)| \leq$$

(in any database D)

Estimating the output size

$$Q = \mathsf{R}_1(\text{?x,?y}) \bowtie \mathsf{R}_2(\text{?y,?z})$$

$$|\mathsf{R}_1| = |\mathsf{R}_2| = n \qquad \Rightarrow \qquad |Q(D)| \leq n^2$$

(in any database D)

Estimating the output size

$$Q = \mathbf{R}_1(?\mathsf{x},?\mathsf{y}) \bowtie \mathbf{R}_2(?\mathsf{y},?\mathsf{z})$$

$$|\mathbf{R}_1| = |\mathbf{R}_2| = n \qquad \Rightarrow \qquad |Q(D)| \leq n^2$$

(in any database D)

$$Q = \mathbf{S}_1(?\mathsf{x},?\mathsf{y}) \bowtie \mathbf{S}_2(?\mathsf{x},?\mathsf{y})$$

$$|\mathbf{S}_1| = |\mathbf{S}_2| = n \qquad \Rightarrow \qquad |Q(D)| \leq$$

(in any database D)

Estimating the output size

$$Q = \mathbf{R}_1(?\mathsf{x},?\mathsf{y}) \bowtie \mathbf{R}_2(?\mathsf{y},?\mathsf{z})$$

$$|\mathbf{R}_1| = |\mathbf{R}_2| = n \qquad \Rightarrow \qquad |Q(D)| \leq n^2$$

(in any database D)

$$Q = \mathbf{S}_1(?\mathsf{x},?\mathsf{y}) \bowtie \mathbf{S}_2(?\mathsf{x},?\mathsf{y})$$

$$|\mathbf{S}_1| = |\mathbf{S}_2| = n \qquad \Rightarrow \qquad |Q(D)| \leq n$$

(in any database D)

Estimating the output size

$$Q = \mathbf{S}_1(\text{?x,?y,?z}) \bowtie \mathbf{S}_2(\text{?y,?z,?w,?x}) \bowtie \mathbf{S}_3(\text{?w,?y,?x})$$

$$\Downarrow$$

$$|Q(D)| \leq$$

(in any database D)

Estimating the output size

$$Q = \mathsf{S}_1(\text{?x,?y,?z}) \bowtie \mathsf{S}_2(\text{?y,?z,?w,?x}) \bowtie \mathsf{S}_3(\text{?w,?y,?x})$$

$$\Downarrow$$

$$|Q(D)| \leq |\mathsf{S}_2^D|$$

(in any database D)

Estimating the output size

$$Q = \mathbf{S}_1(?\mathsf{x},?\mathsf{y},?\mathsf{z}) \bowtie \mathbf{S}_2(?\mathsf{y},?\mathsf{z},?\mathsf{w},?\mathsf{x}) \bowtie \mathbf{S}_3(?\mathsf{w},?\mathsf{y},?\mathsf{x})$$

$$\Downarrow$$

$$|Q(D)| \leq |\mathbf{S}_2^D|$$

(in any database D)

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

$$(\overline{x}_1 \cup \cdots \cup \overline{x}_k) \subseteq \overline{x}_j \qquad \Rightarrow \qquad |Q(D)| \leq |\mathbf{R}_j^D|$$

(in any database D)

Estimating the output size

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

Variables of the query: $(\overline{x}_1 \cup \cdots \cup \overline{x}_k) = \overline{y}$

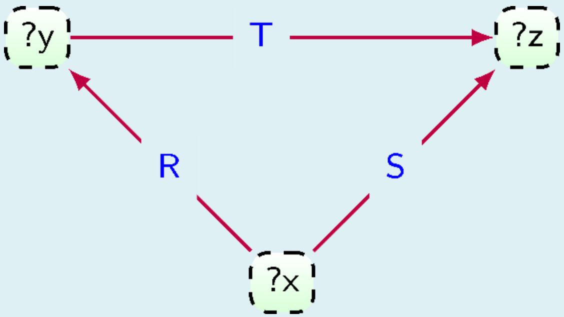
$i_1, \ldots, i_\ell$ s.t. $(\overline{x}_{i_1} \cup \cdots \cup \overline{x}_{i_\ell}) = \overline{y}$

$$\Downarrow$$

$$|Q(D)| \leq \prod_{j=1}^{\ell} |\mathbf{R}_{i_j}^D|$$

(in any database D)

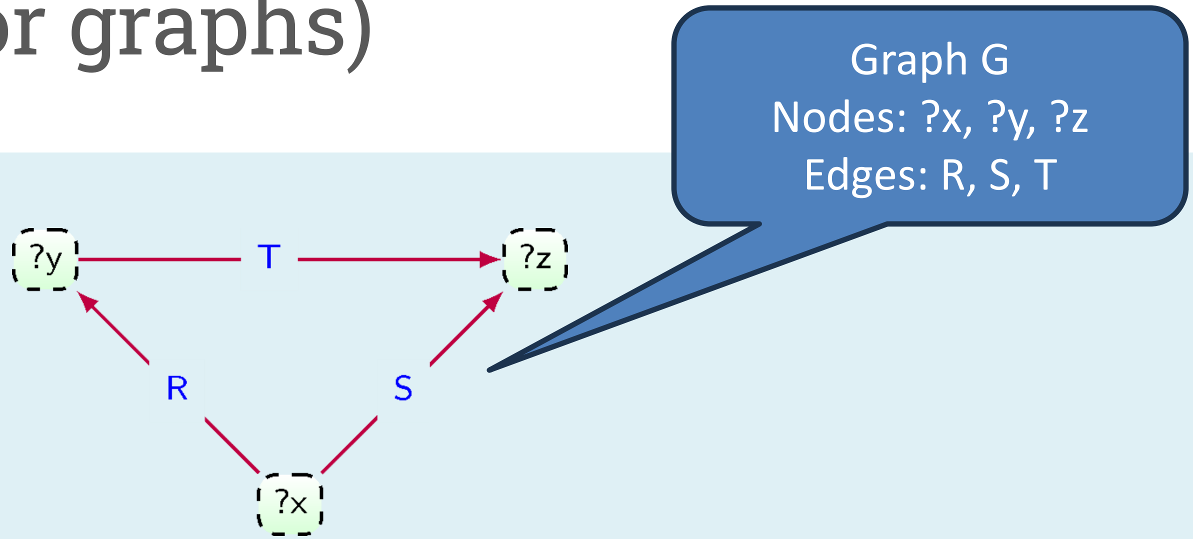
Edge cover (for graphs)



$$Q = \mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$$

- $|\mathbf{output}| \leq |\mathbf{R}| \cdot |\mathbf{S}|$
- $|\mathbf{output}| \leq |\mathbf{R}| \cdot |\mathbf{T}|$
- $|\mathbf{output}| \leq |\mathbf{S}| \cdot |\mathbf{T}|$

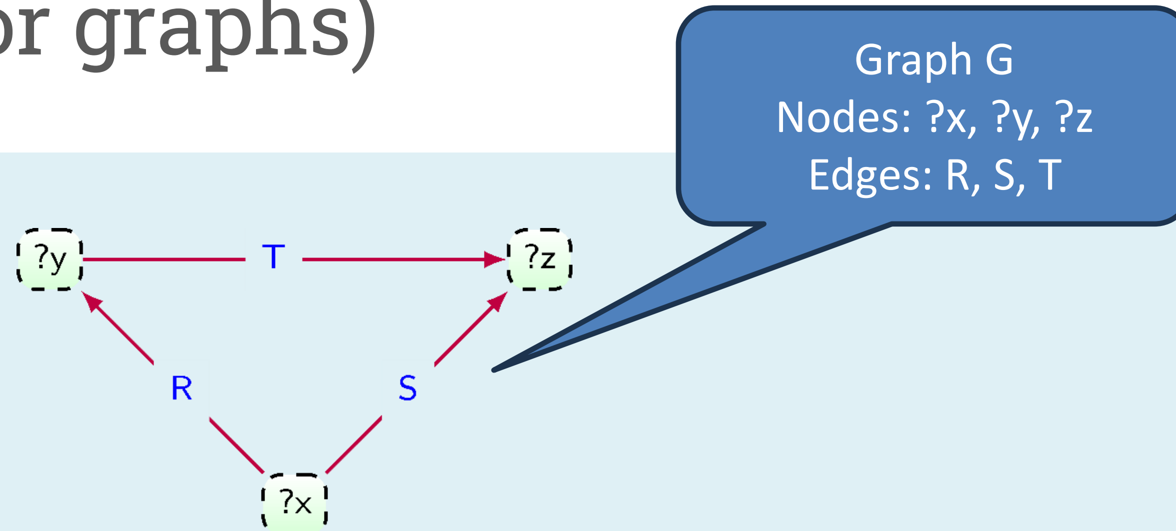
Edge cover (for graphs)



$$Q = \mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$$

- $|\mathbf{output}| \leq |\mathbf{R}| \cdot |\mathbf{S}|$
- $|\mathbf{output}| \leq |\mathbf{R}| \cdot |\mathbf{T}|$
- $|\mathbf{output}| \leq |\mathbf{S}| \cdot |\mathbf{T}|$

Edge cover (for graphs)



$$Q = \mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$$

- $|\mathbf{output}| \leq |\mathbf{R}| \cdot |\mathbf{S}|$ (R, S edge cover)
- $|\mathbf{output}| \leq |\mathbf{R}| \cdot |\mathbf{T}|$ (R, T edge cover)
- $|\mathbf{output}| \leq |\mathbf{S}| \cdot |\mathbf{T}|$ (S, T edge cover)

(in any database D)

Edge cover (for graphs)

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

All $\mathbf{R}_i$ are binary, i.e. $|\overline{x}_i| = 2$

The graph G of Q:

- Nodes: $\overline{x}_1 \cup \overline{x}_2 \cup \cdots \cup \overline{x}_k$
- Edges: $\mathbf{R}_1, \ldots, \mathbf{R}_k$

Edge cover for G:

- Set $\mathbf{R}_{i_1}, \ldots, \mathbf{R}_{i_\ell}$ of edges in G
- S.t. $\overline{x}_{i_1} \cup \cdots \cup \overline{x}_{i_\ell} =$ nodes of G

Edge cover (for graphs)

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

All $\mathbf{R}_i$ are binary, i.e. $|\overline{x}_i| = 2$

The graph G of Q:

- Nodes: $\overline{x}_1 \cup \overline{x}_2 \cup \cdots \cup \overline{x}_k$
- Edges: $\mathbf{R}_1, \ldots, \mathbf{R}_k$

$\Longleftrightarrow$

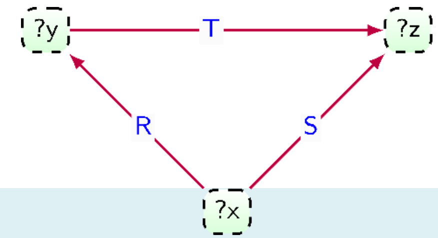
Edge cover for G:

- Set $\mathbf{R}_{i_1}, \ldots, \mathbf{R}_{i_\ell}$ of edges in G
- S.t. $\overline{x}_{i_1} \cup \cdots \cup \overline{x}_{i_\ell} =$ nodes of G

(in any database it holds) $\Downarrow$

$$|\textbf{output}| \leq \prod_{j=1}^{\ell} |\mathbf{R}_{i_j}|$$

Edge cover (another perspective)



	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	1	1	1	0
$?x \xrightarrow{S} ?z$	0	0	0	0
$?y \xrightarrow{T} ?z$	1	0	1	1
	Σ	1	2	1

find: w_R, w_S, w_T

such that:

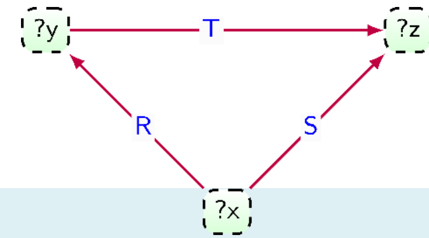
$w_R + w_S \geq 1$ (?x is covered)

$w_R + w_T \geq 1$ (?y is covered)

$w_S + w_T \geq 1$ (?z is covered)

$w_R, w_S, w_T \in \{0, 1\}$

Edge cover (another perspective)



w_R

w_S

w_T

	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	1	1	1	0
$?x \xrightarrow{S} ?z$	0	0	0	0
$?y \xrightarrow{T} ?z$	1	0	1	1
	Σ	1	2	1

find: w_R, w_S, w_T ⟸ edge cover

such that:

$w_R + w_S \geq 1$ (?x is covered)

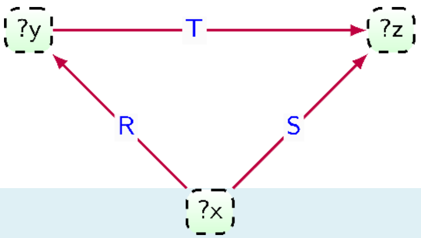
$w_R + w_T \geq 1$ (?y is covered)

$w_S + w_T \geq 1$ (?z is covered)

$w_R, w_S, w_T \in \{0, 1\}$

$$|\textbf{output}| \leq |\textbf{R}|^{w_R} \cdot |\textbf{S}|^{w_S} \cdot |\textbf{T}|^{w_T}$$

Edge cover (we can do one better)



	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	1	1	1	0
$?x \xrightarrow{S} ?z$	0	0	0	0
$?y \xrightarrow{T} ?z$	1	0	1	1
	Σ	1	2	1

find: w_R, w_S, w_T

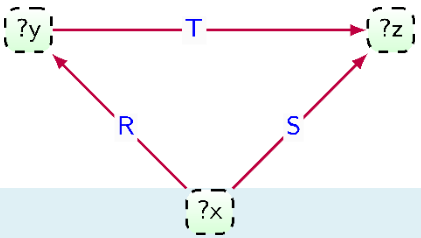
such that: $w_R + w_S \geq 1$ (?x is covered)

$w_R + w_T \geq 1$ (?y is covered)

$w_S + w_T \geq 1$ (?z is covered)

integers $\boxed{w_R, w_S, w_T \in \{0, 1\}}$

Fractional edge cover



	in EC?	?x	?y	?z
?x —R→ ?y	1	1	1	0
?x —S→ ?z	0	0	0	0
?y —T→ ?z	1	0	1	1
	Σ	1	2	1

find: w_R, w_S, w_T

such that:

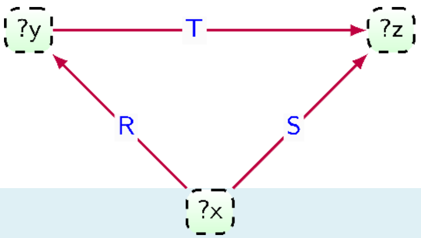
$w_R + w_S \geq 1$ (?x is covered)

$w_R + w_T \geq 1$ (?y is covered)

$w_S + w_T \geq 1$ (?z is covered)

$w_R, w_S, w_T \in [0, 1]$ (rational)

Fractional edge cover



	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
$?x \xrightarrow{S} ?z$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
$?y \xrightarrow{T} ?z$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
	Σ	1	1	1

find: w_R, w_S, w_T

such that:

$w_R + w_S \geq 1$ (?x is covered)

$w_R + w_T \geq 1$ (?y is covered)

$w_S + w_T \geq 1$ (?z is covered)

$w_R, w_S, w_T \in [0, 1]$ (rational)

Fractional edge cover

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

$w_1, \ldots, w_k$ are a **fractional edge cover** for Q if

$$\sum_{R_i : y \in \overline{x}_i} w_i \geq 1 \qquad \text{(for every variable } y\text{)}$$

$$0 \leq w_i \leq 1 \qquad (i = 1, \ldots, k)$$

Intuitively: the fraction allows only some tuples of a relation to participate in the result

AGM bound – upper bound

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

&

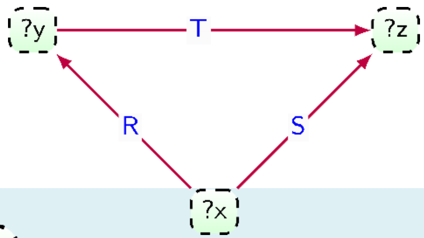
$w_1, \ldots, w_k$ a **fractional edge cover** for Q

$$|Q(D)| \leq |\mathbf{R}_1|^{w_1} \cdot |\mathbf{R}_2|^{w_2} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k}$$

(for any database D; $|\mathbf{R}_i|$ is in D)

See [AGM08] for details

Is the AGM bound tight?



	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
$?x \xrightarrow{S} ?z$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
$?y \xrightarrow{T} ?z$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
	Σ	1	1	1

find: w_R, w_S, w_T

such that:

$w_R + w_S \geq 1$ (?x is covered)

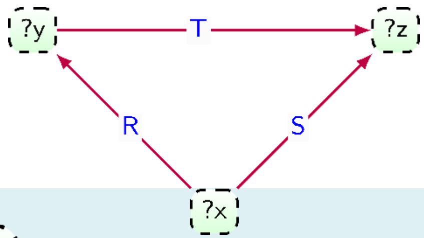
$w_R + w_T \geq 1$ (?y is covered)

$w_S + w_T \geq 1$ (?z is covered)

$w_R, w_S, w_T \in [0, 1]$ (rational)

AGM bound $\Rightarrow$ $|\textbf{output}| \leq |\textbf{R}|^{w_R} \cdot |\textbf{S}|^{w_S} \cdot |\textbf{T}|^{w_T}$

Is the AGM bound tight?



	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
$?x \xrightarrow{S} ?z$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
$?y \xrightarrow{T} ?z$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
	Σ	1	1	1

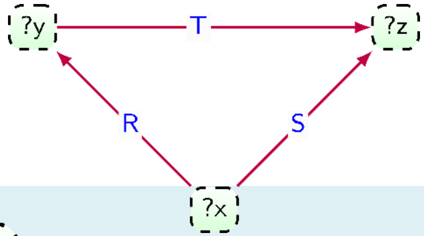
(for any database with fixed $|\mathbf{R}| = n_R, |\mathbf{S}| = n_S, |\mathbf{T}| = n_T$)

find: w_R, w_S, w_T

such that:
$w_R + w_S \geq 1$ (?x is covered)
$w_R + w_T \geq 1$ (?y is covered)
$w_S + w_T \geq 1$ (?z is covered)
$w_R, w_S, w_T \in [0, 1]$ (rational)

AGM bound $\Rightarrow$ $|\textbf{output}| \leq |\mathbf{R}|^{w_R} \cdot |\mathbf{S}|^{w_S} \cdot |\mathbf{T}|^{w_T}$

Is the AGM bound tight?



	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
$?x \xrightarrow{S} ?z$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
$?y \xrightarrow{T} ?z$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
	Σ	1	1	1

(for any database with fixed $|\mathbf{R}| = n_R, |\mathbf{S}| = n_S, |\mathbf{T}| = n_T$)

find: w_R, w_S, w_T

such that: $w_R + w_S \geq 1$ (?x is covered)

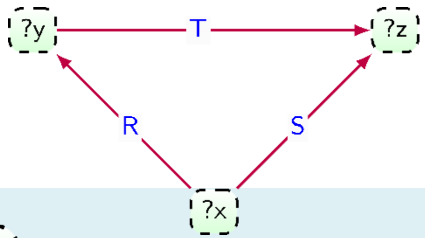
$w_R + w_T \geq 1$ (?y is covered)

$w_S + w_T \geq 1$ (?z is covered)

$w_R, w_S, w_T \in [0, 1]$ (rational)

AGM bound $\Rightarrow$ $|\textbf{output}| \leq n_R^{w_R} \cdot n_S^{w_S} \cdot n_T^{w_T}$

Is the AGM bound tight?



	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
$?x \xrightarrow{S} ?z$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
$?y \xrightarrow{T} ?z$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
	Σ	1	1	1

(for any database with fixed $|\mathbf{R}| = n_R, |\mathbf{S}| = n_S, |\mathbf{T}| = n_T$)

minimize: $n_R^{w_R} \cdot n_S^{w_S} \cdot n_T^{w_T}$

such that:
$$w_R + w_S \geq 1$$
$$w_R + w_T \geq 1$$
$$w_S + w_T \geq 1$$
$$w_R, w_S, w_T \in [0, 1]$$

w_R^m, w_S^m, w_T^m optimal solution

$\Downarrow$

$$|\textbf{output}| \leq n_R^{w_R} \cdot n_S^{w_S} \cdot n_T^{w_T}$$

on any database with

$|\mathbf{R}| = n_R, |\mathbf{S}| = n_S, |\mathbf{T}| = n_T$

Is the AGM bound tight?

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \ \mathbf{R}_2(\overline{x}_2) \bowtie \ \cdots \ \bowtie \mathbf{R}_k(\overline{x}_k)$$

Consider only instances D s.t. $|\mathbf{R}_i| = n_i$

$w_1, \ldots, w_k$ **fractional edge cover**:

$\Downarrow$ (for any instance D)

$$|Q(D)| \leq |\mathbf{R}_1|^{w_1} \cdot |\mathbf{R}_2|^{w_2} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k}$$

Is the AGM bound tight?

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

Consider only instances D s.t. $|\mathbf{R}_i| = n_i$

$w_1, \ldots, w_k$ **fractional edge cover**:

$$\Downarrow \quad \text{(for any instance } D \text{ s.t. } |\mathbf{R}_i| = n_i\text{)}$$

$$|Q(D)| \leq n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k}$$

Is the AGM bound tight?

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \ \mathbf{R}_2(\overline{x}_2) \bowtie \ \cdots \ \bowtie \mathbf{R}_k(\overline{x}_k)$$

Consider only instances D s.t. $|\mathbf{R}_i| = n_i$

$w_1, \ldots, w_k$ **fractional edge cover**:

(for any instance D s.t. $|\mathbf{R}_i| = n_i$)

$$|Q(D)| \leq n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k}$$

We can find the best fractional edge cover over *all* such instances!

Is the AGM bound tight?

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

Consider only instances D s.t. $|\mathbf{R}_i| = n_i$

minimize: $n_1^{w_1} \cdot n_2^{w_2} \cdot \dots \cdot n_k^{w_k}$

such that: $\sum_{R_i : y \in \overline{x}_i} w_i \geq 1$ (for every variable y)

$w_i \in [0,1]$ $(i = 1, \dots, k)$

$|Q(D)| \leq n_1^{w_1^m} \cdot n_2^{w_2^m} \cdot \dots \cdot n_k^{w_k^m}$, for $w_1^m, \dots, w_k^m$ optimal solution

(over all such instances D)

AGM bound – lower bound

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

assume $|\mathbf{R}_i| = n_i$, where $n_1, \ldots, n_k$ are fixed

$w_1^m, \ldots, w_k^m$ is a **solution for the LP**:

$$\begin{array}{lll} \text{minimize:} & n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k} & \\ \text{such that:} & \displaystyle\sum_{R_i : y \in \overline{x}_i} w_i \geq 1 & \text{(for every variable } y\text{)} \\ & 0 \leq w_i \leq 1 & (i = 1, \ldots, k) \end{array}$$

There exists a database D where:

$$|Q(D)| = |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$$

See [AGM08] for details

AGM bound – recap

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

assume $|\mathbf{R}_i| = n_i$, where $n_1, \ldots, n_k$ are fixed

$w_1^m, \ldots, w_k^m$ is a **solution for the LP**:

minimize: $\quad n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k}$

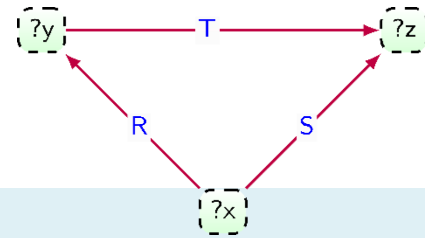
such that: $\quad \sum_{R_i : y \in \overline{x}_i} w_i \geq 1 \quad$ (for every variable y)

$\quad 0 \leq w_i \leq 1 \qquad (i = 1, \ldots, k)$

- $|Q(D)| \leq |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (for all such D)
- $|Q(D)| = |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (on one such D)

See [AGM08] for details

For our motivating query



	in EC?	?x	?y	?z
$?x \xrightarrow{R} ?y$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
$?x \xrightarrow{S} ?z$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
$?y \xrightarrow{T} ?z$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
	Σ	1	1	1

minimize: $n_R^{w_R} \cdot n_S^{w_S} \cdot n_T^{w_T}$

such that: $w_R + w_S \geq 1$

$w_R + w_T \geq 1$

$w_S + w_T \geq 1$

$w_R, w_S, w_T \in [0, 1]$

$\frac{1}{2}, \frac{1}{2}, \frac{1}{2}$ optimal solution

$$|\textbf{output}| \leq \sqrt{n_R \cdot n_S \cdot n_T}$$

on any database with

$|\mathbf{R}| = n_R, |\mathbf{S}| = n_S, |\mathbf{T}| = n_T$

Hyperedge cover (general AGM bound)

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

($\mathbf{R}_i$ are of any arity)

The hypergraph G of Q:

- Nodes: $\overline{x}_1 \cup \overline{x}_2 \cup \cdots \cup \overline{x}_k$
- Hyperedges: $\mathbf{R}_1, \ldots, \mathbf{R}_k$

$\Longleftrightarrow$

Hyperedge cover for G:

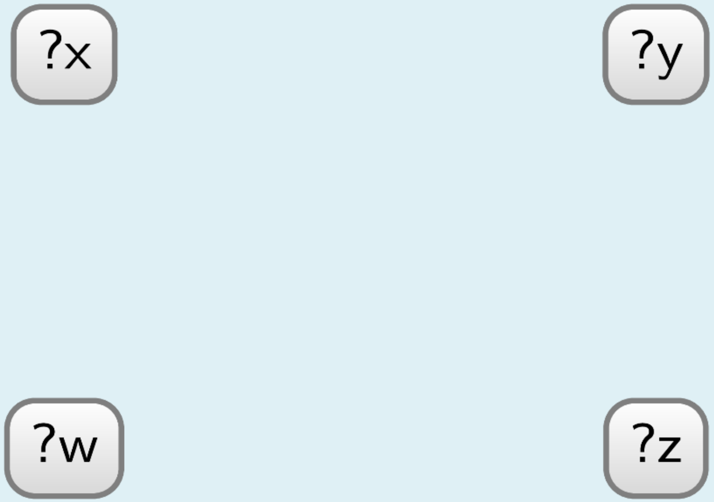
- Set $\mathbf{R}_{i_1}, \ldots, \mathbf{R}_{i_\ell}$ of hyperedges in G
- S.t. $\overline{x}_{i_1} \cup \cdots \cup \overline{x}_{i_\ell}$ = nodes of G

(in any database it holds) $\Downarrow$

$$|\mathbf{output}| \leq \prod_{j=1}^{\ell} |\mathbf{R}_{i_j}|$$

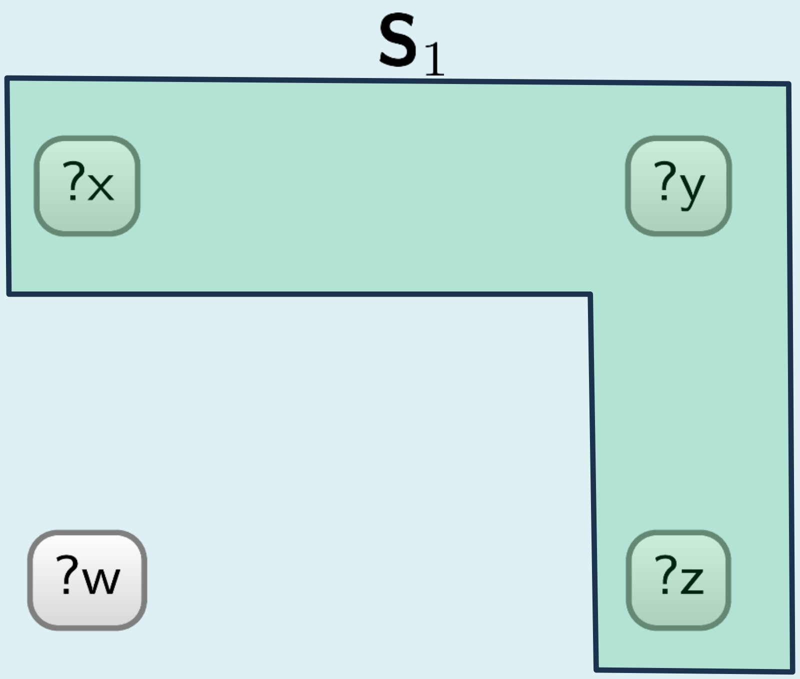
Hyperedge cover (for relations)

$$\mathbf{S}_1(?\mathrm{x},?\mathrm{y},?\mathrm{z}) \bowtie \mathbf{S}_2(?\mathrm{y},?\mathrm{z},?\mathrm{w}) \bowtie \mathbf{S}_3(?\mathrm{w},?\mathrm{z})$$



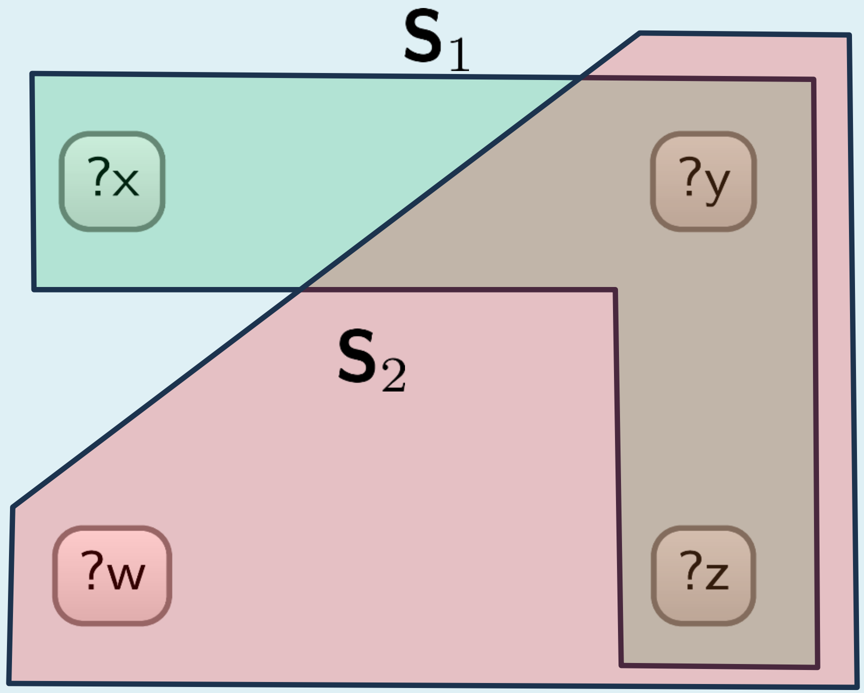
Hyperedge cover (for relations)

$$\mathbf{S}_1(?x,?y,?z) \bowtie \mathbf{S}_2(?y,?z,?w) \bowtie \mathbf{S}_3(?w,?z)$$



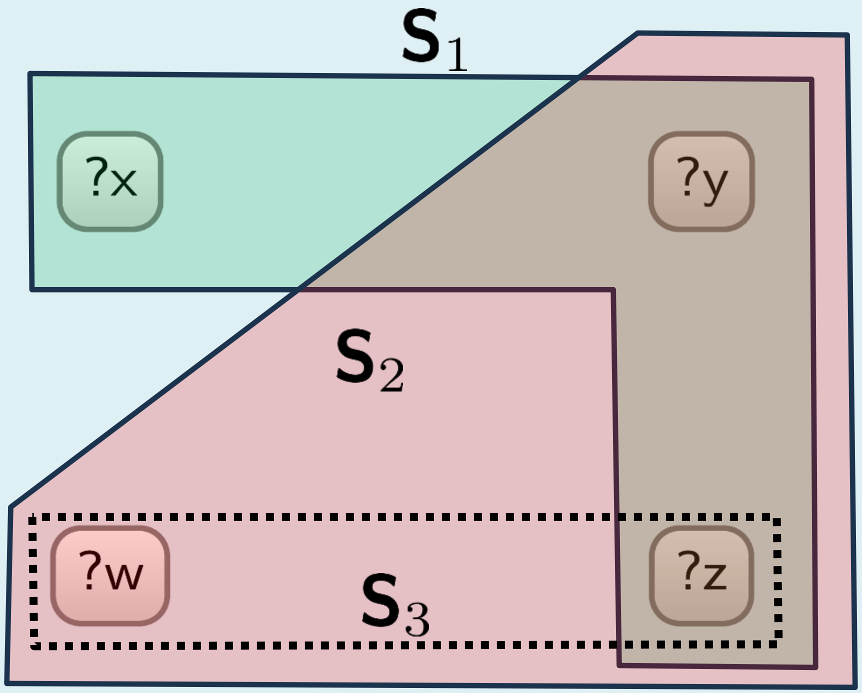
Hyperedge cover (for relations)

$$\mathbf{S}_1(?x,?y,?z) \bowtie \mathbf{S}_2(?y,?z,?w) \bowtie \mathbf{S}_3(?w,?z)$$



Hyperedge cover (for relations)

$$\mathbf{S}_1(?x,?y,?z) \bowtie \mathbf{S}_2(?y,?z,?w) \bowtie \mathbf{S}_3(?w,?z)$$



Hyperedge covers:

$\mathbf{S}_1, \mathbf{S}_2$

$\mathbf{S}_1, \mathbf{S}_3$

Worst-case optimal algorithms

- Best possible algorithm for a query Q :
  - $O(1)$ per query results
  - So runtime would be $O(|Q(D)|)$ on any instance D
  - This is the holy grail of databases!
    - So it probably does not exist

- Something more realistic:
  - Join query: $Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$
  - I give you any instance where $|\mathbf{R}_i| = n_i$
  - The algorithm runs the best it can on any such instance

What does the ''best it can'' mean?

Worst-case optimal algorithms

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

assume $|\mathbf{R}_i| = n_i$, where $n_1, \ldots, n_k$ are fixed

$w_1^m, \ldots, w_k^m$ is a **solution for the LP**:

$$\begin{array}{lll} \text{minimize:} & n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k} & \\ \text{such that:} & \displaystyle\sum_{R_i : y \in \overline{x}_i} w_i \geq 1 & \text{(for every variable } y\text{)} \\ & 0 \leq w_i \leq 1 & (i = 1, \ldots, k) \end{array}$$

- $|Q(D)| \leq |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (for all such D)
- $|Q(D)| = |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (on one such D)

See [AGM08] for details

Worst-case optimal algorithms

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

assume $|\mathbf{R}_i| = n_i$, where $n_1, \ldots, n_k$ are fixed

$w_1^m, \ldots, w_k^m$ is a **solution for the LP**:

minimize: $\quad n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k}$

You cannot be worse than this!

$1 \quad$ (for every variable y)

$\leq 1 \quad (i = 1, \ldots, k)$

- $|Q(D)| \leq |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (for all such D)
- $|Q(D)| = |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (on one such D)

See [AGM08] for details

Worst-case optimal algorithms

$$Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$$

assume $|\mathbf{R}_i| = n_i$, where $n_1, \dots, n_k$ are fixed

$w_1^m, \dots, w_k^m$ is a **solu**

It can actually be this bad!

minimize: $n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k}$

1 (fo variable y)

You cannot be worse than this!

≤ 1 $(i$ $\dots, k)$

- $|Q(D)| \leq |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdot$ $\cdot\, |\mathbf{R}_k|^{w_k^m}$ (for all such D)
- $|Q(D)| = |\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m}$ (on one such D)

See [AGM08] for details

Worst-case optimal algorithms

a join algorithm is **worst-case optimal** if

for any $Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$

it runs in $O(|\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m})$

on any instance D with $|\mathbf{R}_i| = n_i$

where $w_1^m, \ldots, w_k^m$ is a **solution for the LP**:

minimize: $n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k}$

such that: $\sum_{R_i : y \in \overline{x}_i} w_i \geq 1$ (for every variable y)

$0 \leq w_i \leq 1 \quad (i = 1, \ldots, k)$

See [Ngo13] for details

Worst-case optimal algorith

Up to a logarithmic factor!

a join algorithm is **worst-case optimal** if

for any $Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$

it runs in $O(|\mathbf{R}_1|^{w_1^m} \cdot |\mathbf{R}_2|^{w_2^m} \cdot \cdots \cdot |\mathbf{R}_k|^{w_k^m})$

on any instance D with $|\mathbf{R}_i| = n_i$

where $w_1^m, \ldots, w_k^m$ is a **solution for the LP**:

$$
\begin{array}{lll}
\text{minimize:} & n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k} & \\
\text{such that:} & \displaystyle\sum_{R_i : y \in \overline{x}_i} w_i \geq 1 & \text{(for every variable } y\text{)} \\
 & 0 \leq w_i \leq 1 & (i = 1, \ldots, k)
\end{array}
$$

See [Ngo13] for details

Worst-case optimal algorithms

$$AGM(Q, D)$$

for $Q = \mathbf{R}_1(\overline{x}_1) \bowtie \mathbf{R}_2(\overline{x}_2) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x}_k)$

D an instance with $|\mathbf{R}_i| = n_i$

is the value $n_1^{w_1^m} \cdot n_2^{w_2^m} \cdot \cdots \cdot n_k^{w_k^m}$

where $w_1^m, \ldots, w_k^m$ is a **solution for the LP**:

$$\begin{array}{lll} \text{minimize:} & n_1^{w_1} \cdot n_2^{w_2} \cdot \cdots \cdot n_k^{w_k} & \\ \text{such that:} & \displaystyle\sum_{R_i : y \in \overline{x}_i} w_i \geq 1 & \text{(for every variable } y\text{)} \\ & 0 \leq w_i \leq 1 & (i = 1, \ldots, k) \end{array}$$

See [Ngo13] for details

Worst-case optimal algorithms

a join algorithm is **worst-case optimal** if

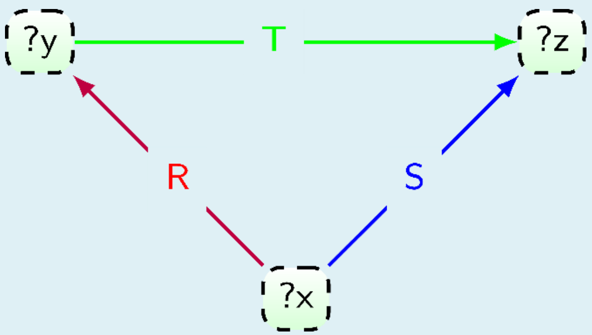
it runs in time $O(AGM(Q, D))$

for any join query Q and a database D

(up to a logarithmic factor)

See [Ngo13] for details

Are pairwise joins wco?

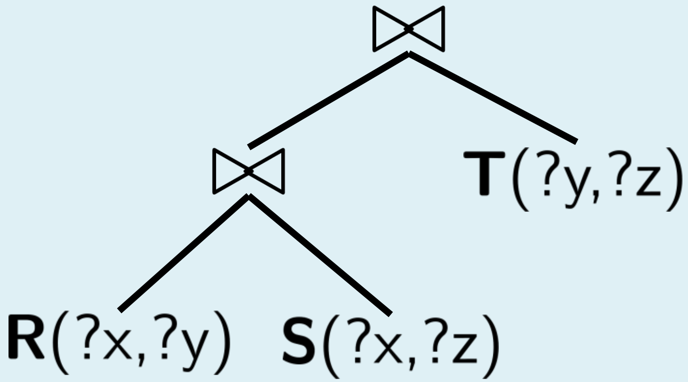


$$Q = \mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$$

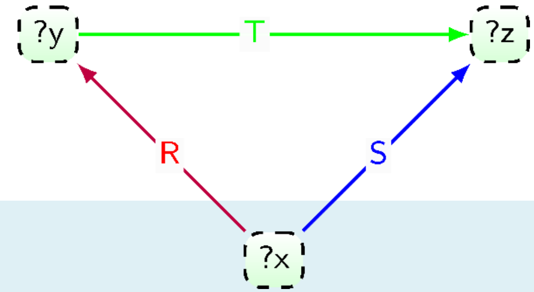
$$|Q(D)| \leq n^{\frac{3}{2}}$$ **(AGM bound)**

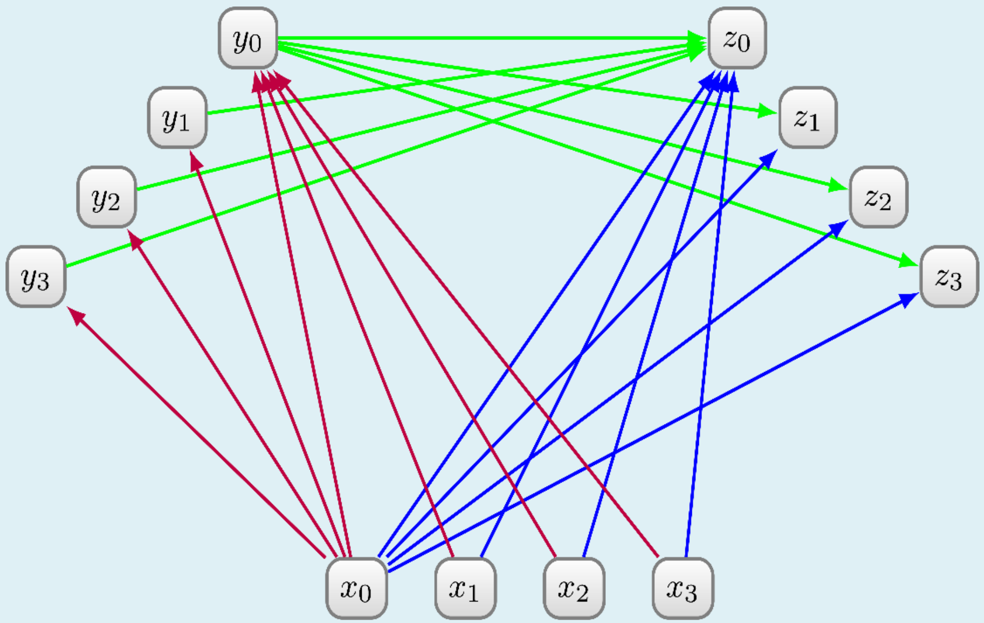
on any database with $|\mathbf{R}| = |\mathbf{S}| = |\mathbf{T}| = n$

Maybe we can find a good ordering?



Are pairwise joins wco?





$$\mathbf{R} = \{x_0\} \times \{y_0, \ldots, y_m\} \cup \{x_0, \ldots x_m\} \times \{y_0\}$$

$$\mathbf{S} = \{x_0\} \times \{z_0, \ldots, z_m\} \cup \{x_0, \ldots x_m\} \times \{z_0\}$$

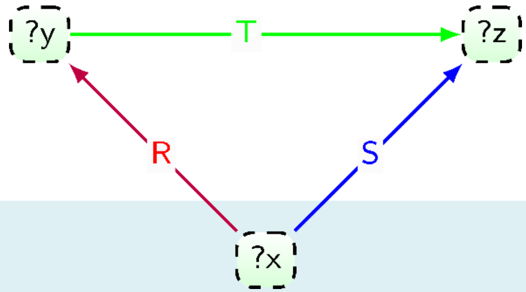
$$\mathbf{T} = \{y_0\} \times \{z_0, \ldots, z_m\} \cup \{y_0, \ldots y_m\} \times \{z_0\}$$

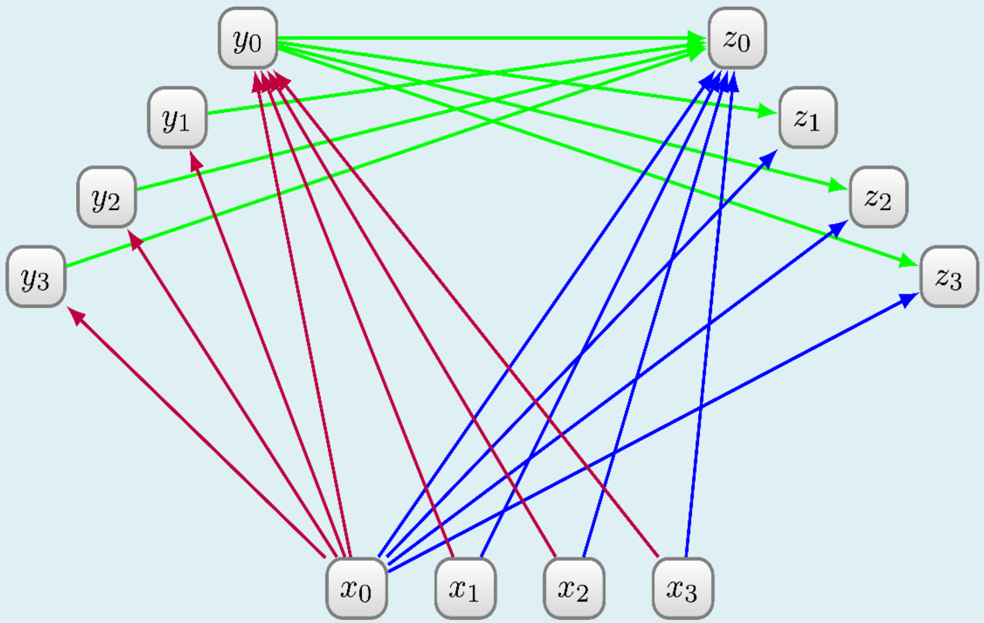
Observations:

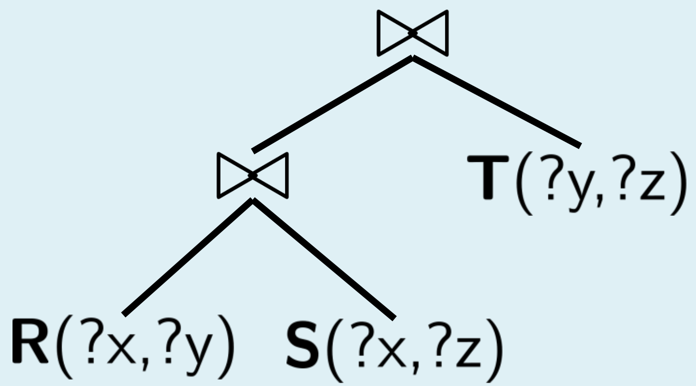
$$|\mathbf{R}| = |\mathbf{S}| = |\mathbf{T}| = 2m + 1 \quad \Rightarrow \quad AGM(Q, D) = (2m+1)^{\frac{3}{2}}$$

$$|\mathbf{R} \bowtie \mathbf{S}| = |\mathbf{R} \bowtie \mathbf{T}| = |\mathbf{S} \bowtie \mathbf{T}| = m^2 + m$$

Are pairwise joins wco?





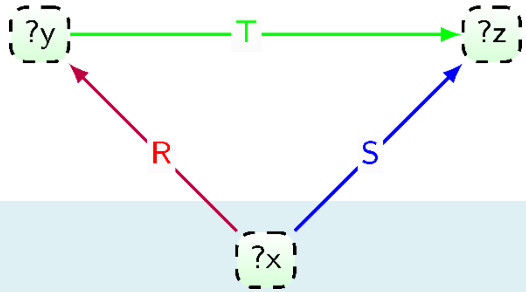


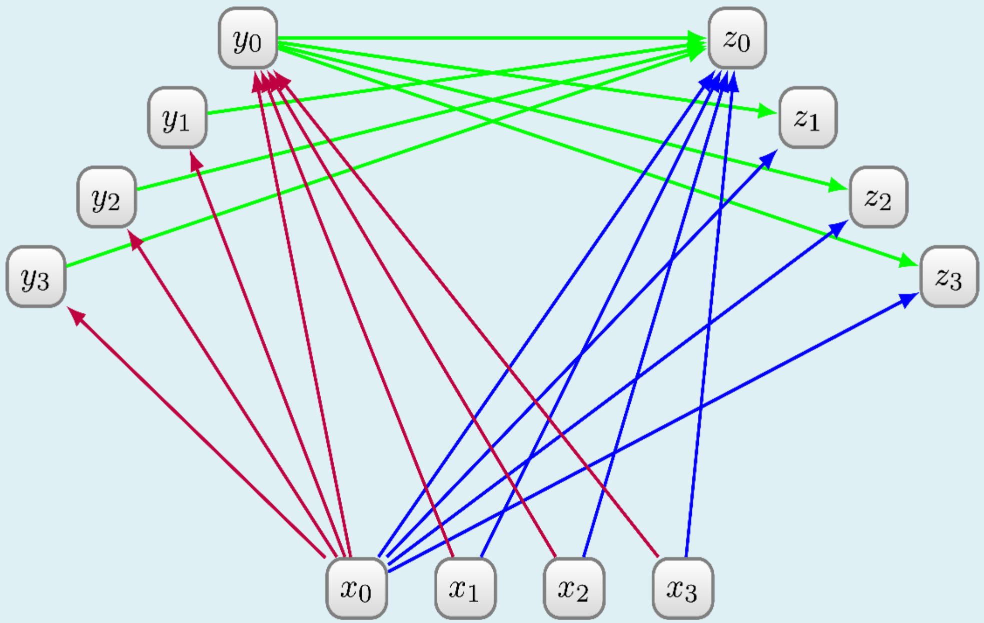
Observations:

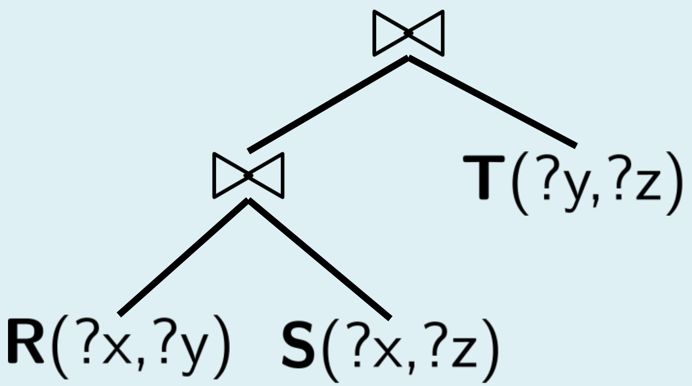
$$|\mathbf{R}| = |\mathbf{S}| = |\mathbf{T}| = 2m+1 \quad \Rightarrow \quad AGM(Q,D) = (2m+1)^{\frac{3}{2}}$$

$$|\mathbf{R} \bowtie \mathbf{S}| = |\mathbf{R} \bowtie \mathbf{T}| = |\mathbf{S} \bowtie \mathbf{T}| = m^2 + m$$

Are pairwise joins wco?







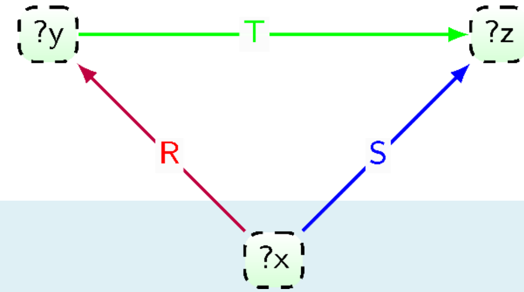
$$O(m^2)$$

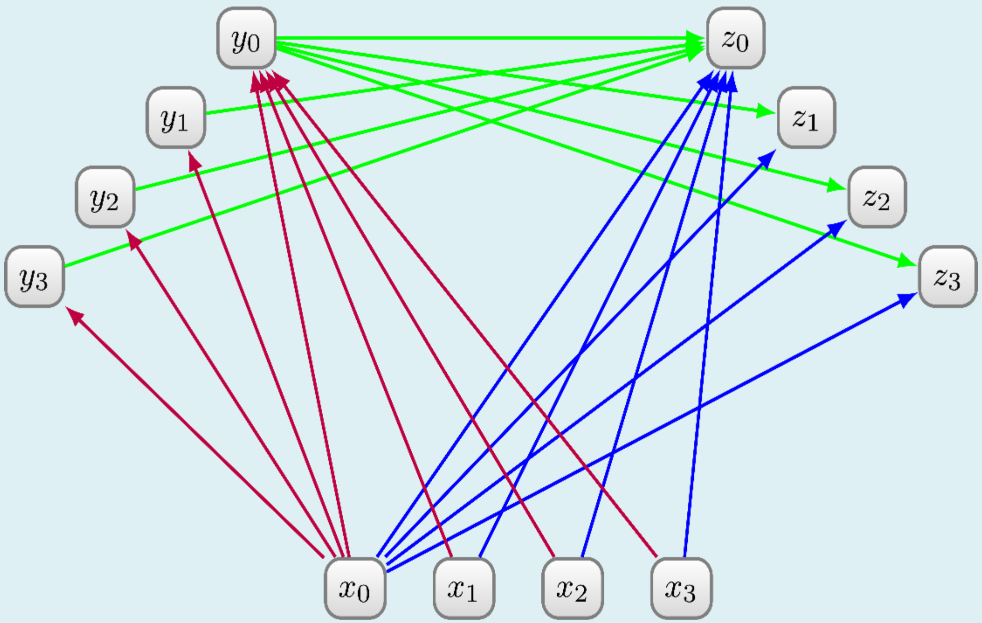
Observations:

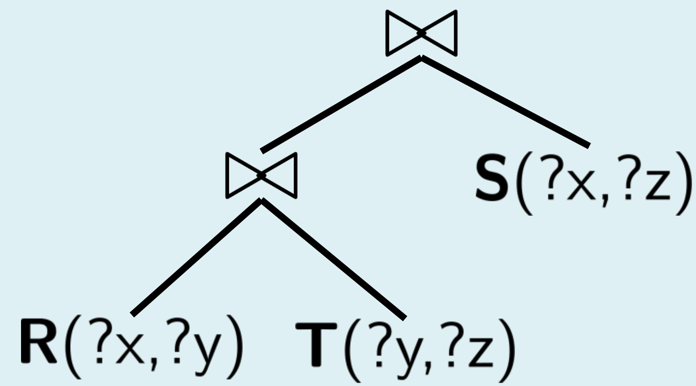
$$|\mathbf{R}| = |\mathbf{S}| = |\mathbf{T}| = 2m+1 \quad \Rightarrow \quad AGM(Q,D) = (2m+1)^{\frac{3}{2}}$$

$$|\mathbf{R} \bowtie \mathbf{S}| = |\mathbf{R} \bowtie \mathbf{T}| = |\mathbf{S} \bowtie \mathbf{T}| = m^2 + m$$

Are pairwise joins wco?







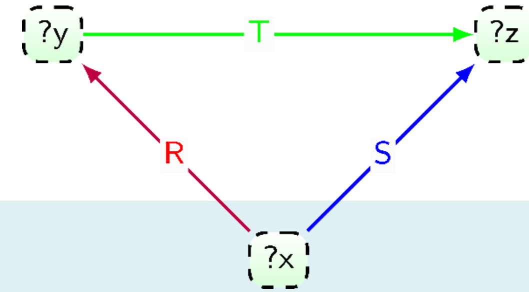
$$O(m^2)$$

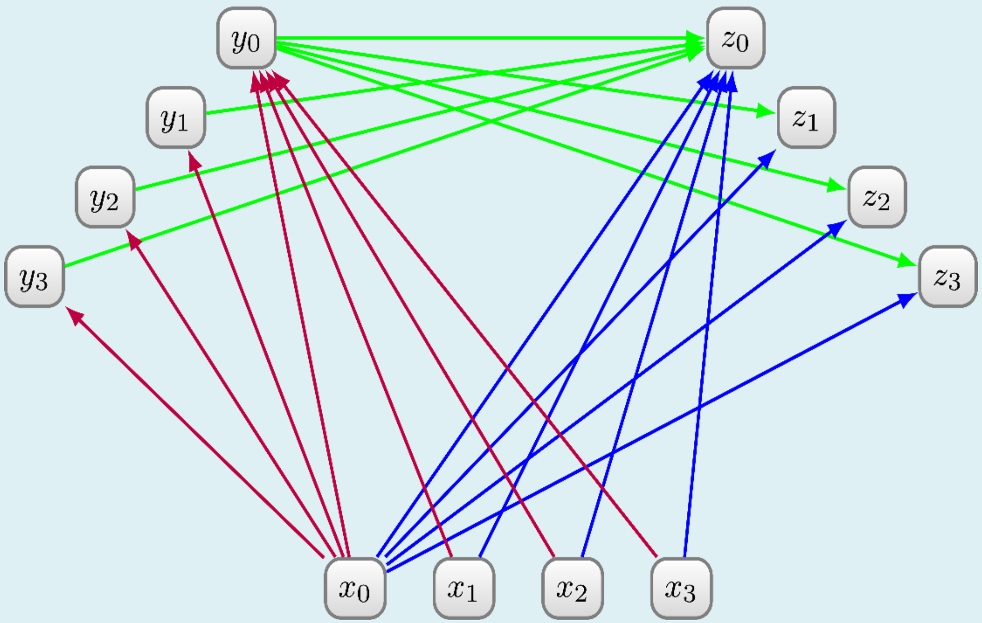
Observations:

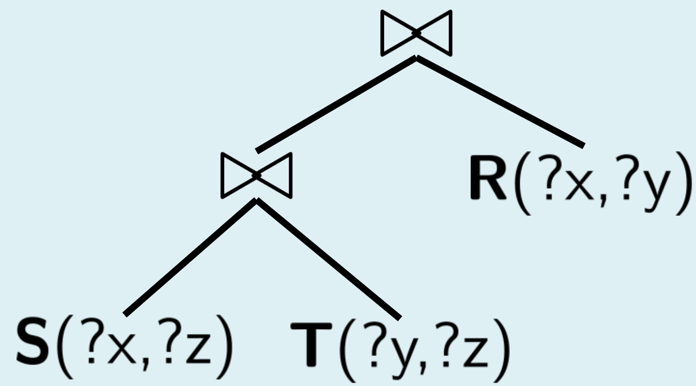
$$|\mathbf{R}| = |\mathbf{S}| = |\mathbf{T}| = 2m + 1 \quad \Rightarrow \quad AGM(Q, D) = (2m+1)^{\frac{3}{2}}$$

$$|\mathbf{R} \bowtie \mathbf{S}| = |\mathbf{R} \bowtie \mathbf{T}| = |\mathbf{S} \bowtie \mathbf{T}| = m^2 + m$$

Are pairwise joins wco?







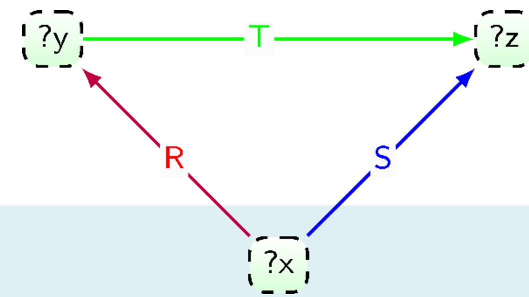
$O(m^2)$

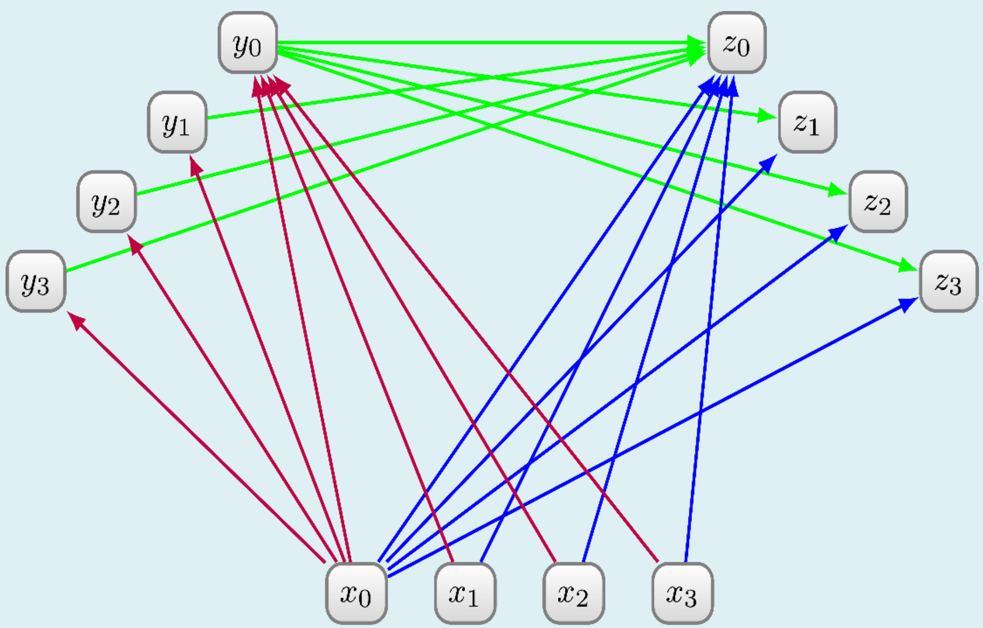
Observations:

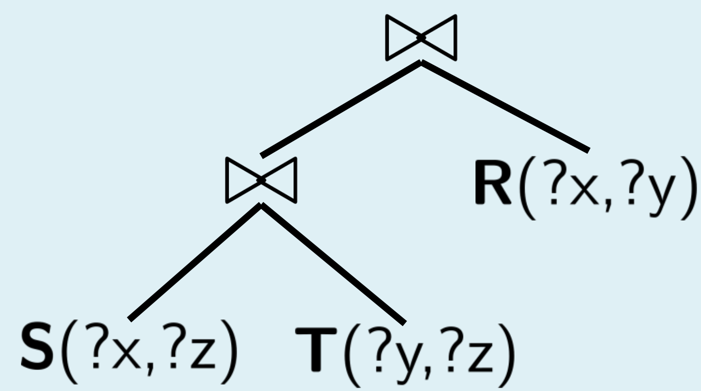
$$|\mathbf{R}| = |\mathbf{S}| = |\mathbf{T}| = 2m + 1 \quad \Rightarrow \quad AGM(Q, D) = (2m+1)^{\frac{3}{2}}$$

$$|\mathbf{R} \bowtie \mathbf{S}| = |\mathbf{R} \bowtie \mathbf{T}| = |\mathbf{S} \bowtie \mathbf{T}| = m^2 + m$$

Are pairwise joins wco?







$O(m^2)$

Conclusion:

Pairwise joins are not worst-case optimal!

Example of a WCO algorithm: Leapfrog Triejoin

Unary joins

$$Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \cdots \bowtie \mathbf{R}_{n-1}(?x)$$

Relations stored in **increasing order**

$$\left.\begin{array}{l}
\bullet\ \mathbf{R}_i.\text{begin}() : \text{get } \textit{before} \text{ the first value} \\
\bullet\ \mathbf{R}_i.\text{key}() : \text{return the value at current position} \\
\bullet\ \mathbf{R}_i.\text{next}() : \text{advance to the next position}
\end{array}\right\} O(1)$$

$$\left.\bullet\ \mathbf{R}_i.\text{seek}(k) : \text{advance to first element} \geq k \right\} O(log|\mathbf{R}|)$$

Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$

$\mathbf{R}_0$: $\xrightarrow{\text{next()}}$ 1		3	4	5	6	7	8		10		12
$\mathbf{R}_1$:	2				6	7	8	9			12
$\mathbf{R}_2$:	2		4	5			8		10	11	12

Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$

$\mathbf{R}_0$: $\xrightarrow{\text{next()}}$	1		3	4	5	6	7	8		10		12
$\mathbf{R}_1$: $\xrightarrow{\text{seek(1)}}$		2				6	7	8	9			12
$\mathbf{R}_2$:		2		4	5			8		10	11	12

Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

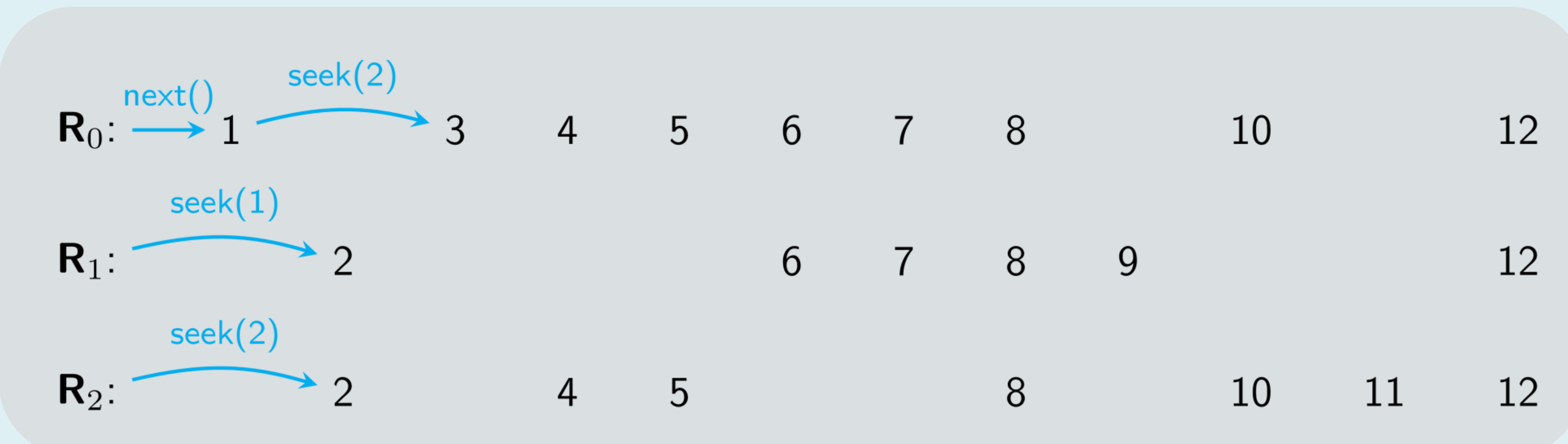
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$

$\mathbf{R}_0$: $\xrightarrow{\text{next()}}$	1		3	4	5	6	7	8		10		12
$\mathbf{R}_1$: $\xrightarrow{\text{seek(1)}}$		2				6	7	8	9			12
$\mathbf{R}_2$: $\xrightarrow{\text{seek(2)}}$		2		4	5			8		10	11	12

Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

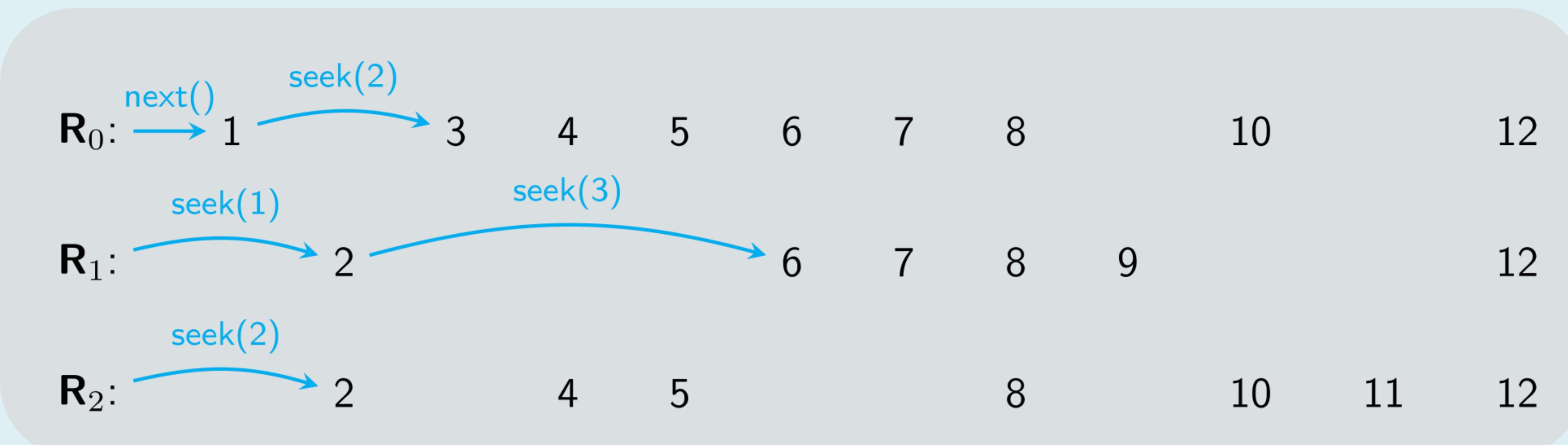
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

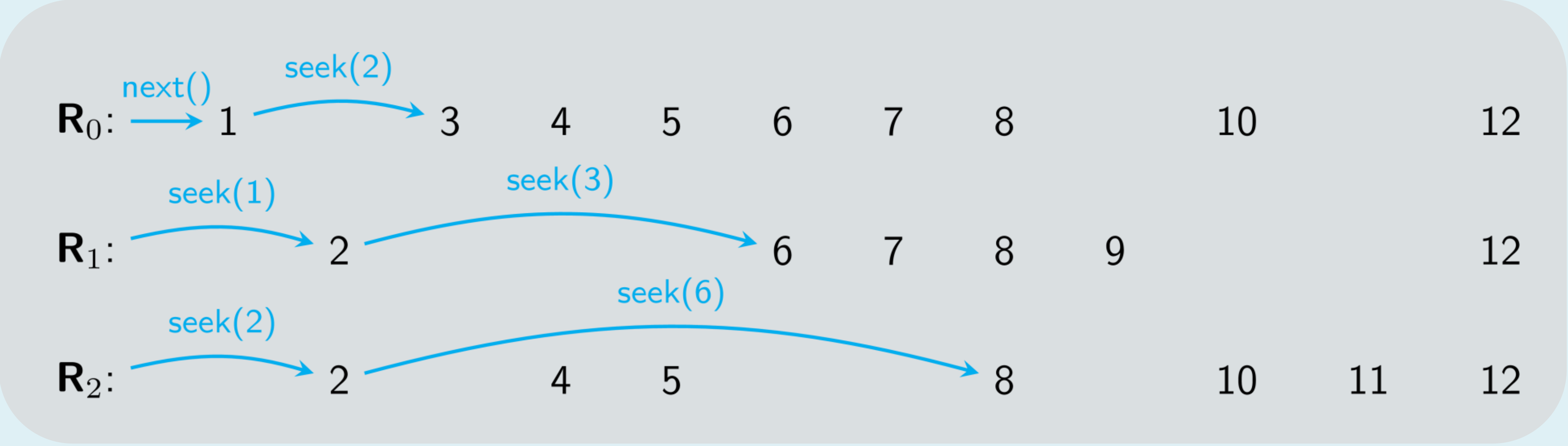
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

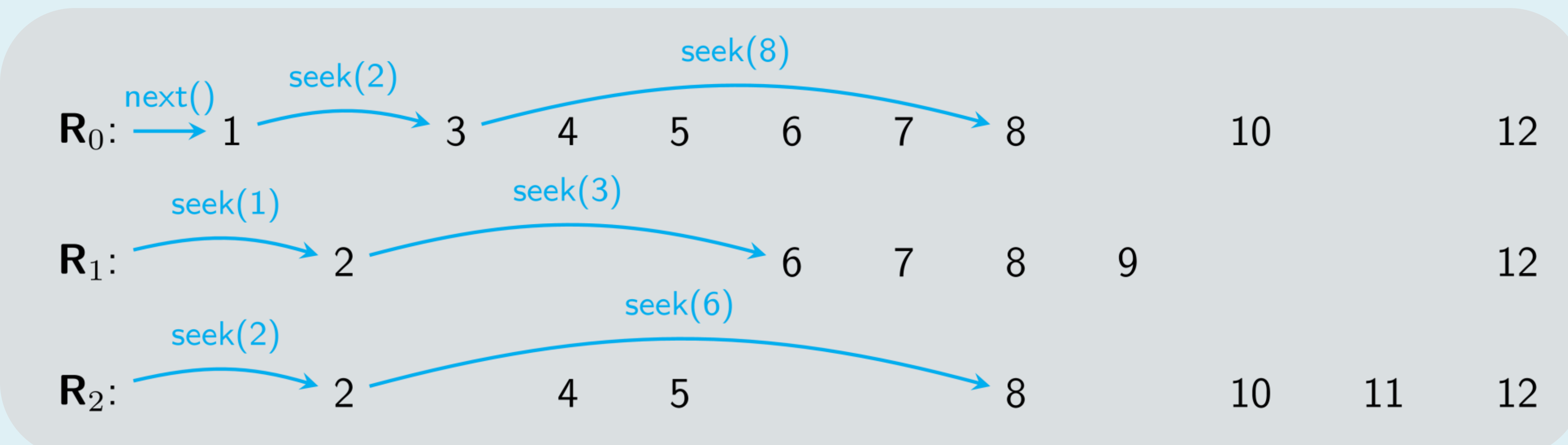
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

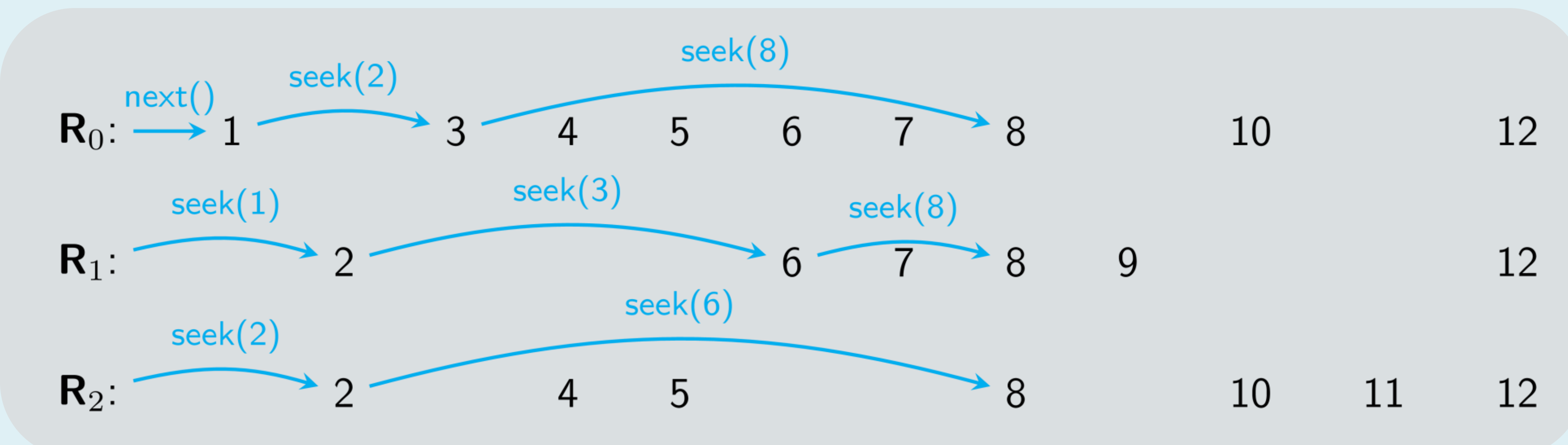
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

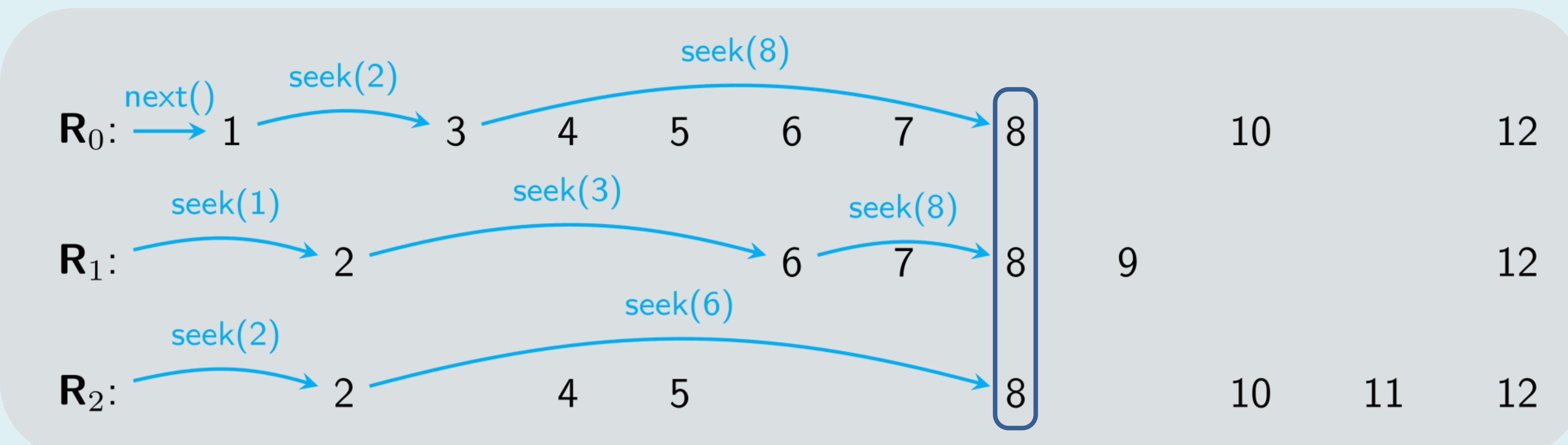
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

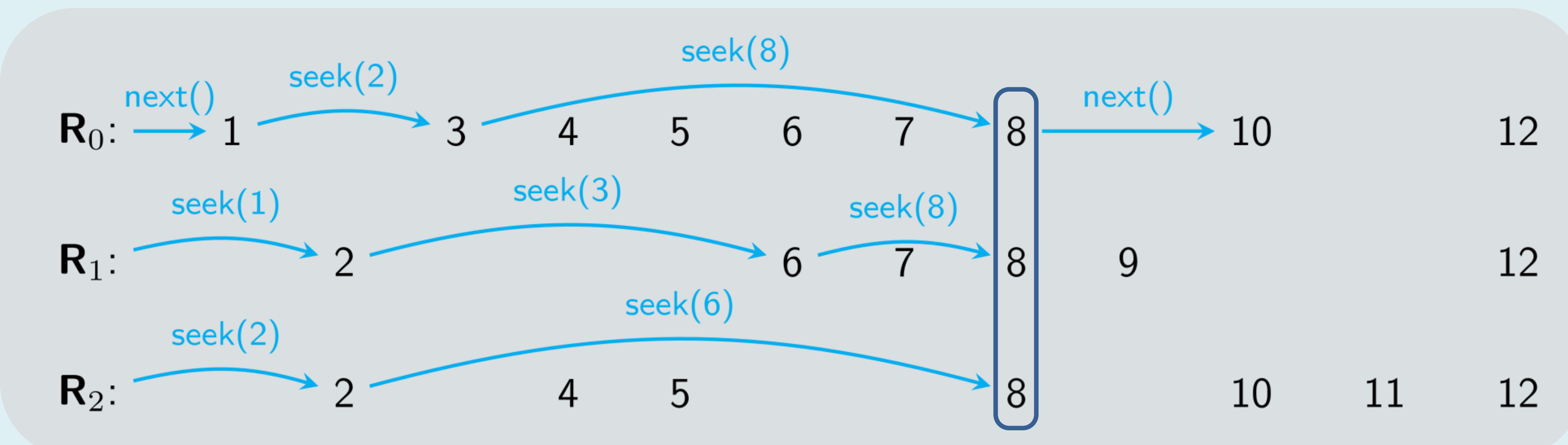
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

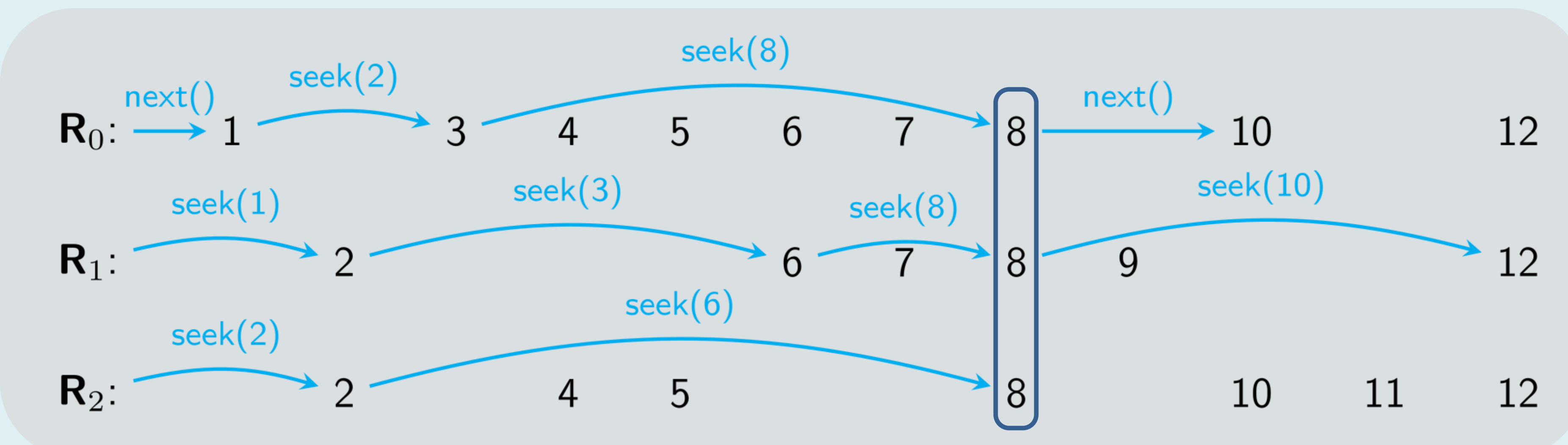
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

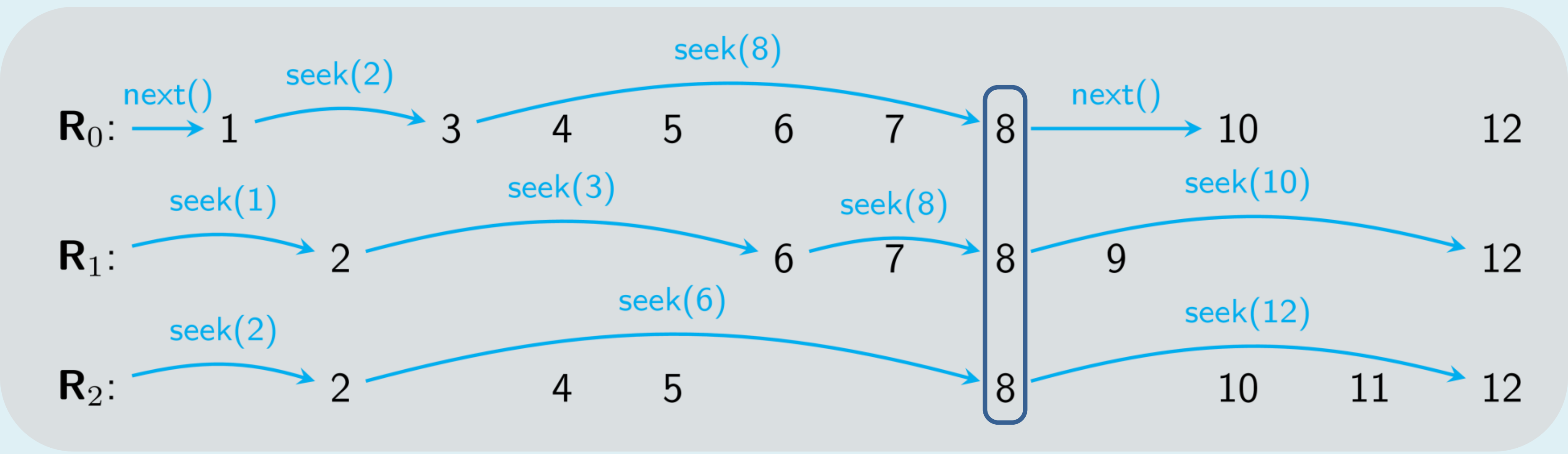
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

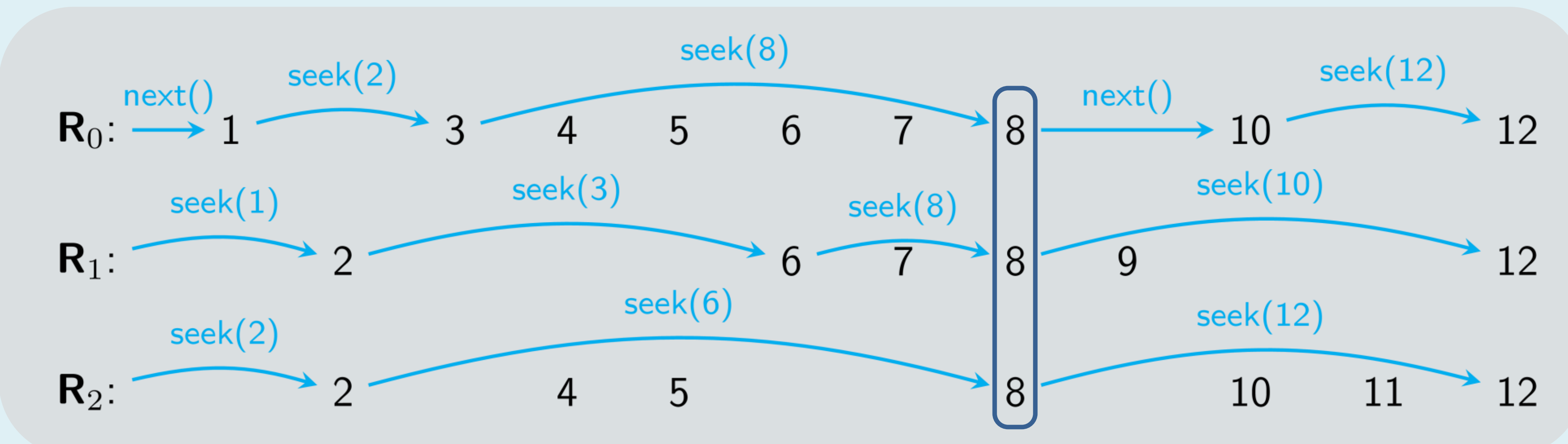
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

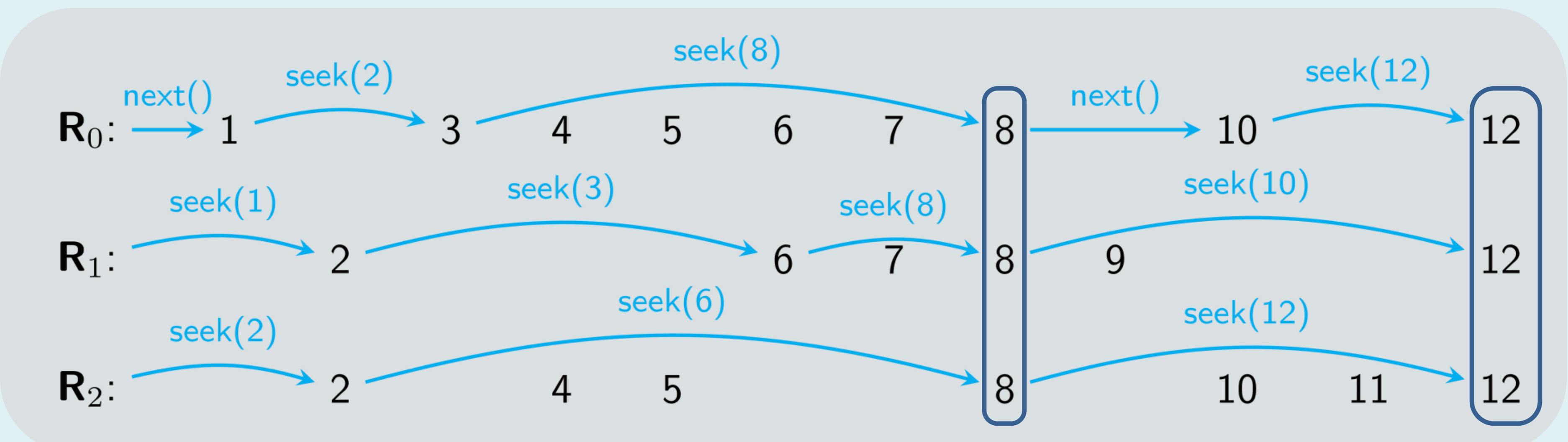
Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

Evaluate $Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \mathbf{R}_2(?x)$



Unary joins

$$Q = \mathbf{R}_0(?x) \bowtie \mathbf{R}_1(?x) \bowtie \cdots \bowtie \mathbf{R}_{n-1}(?x)$$

Relations stored in **increasing order**

Leapfrog-next():

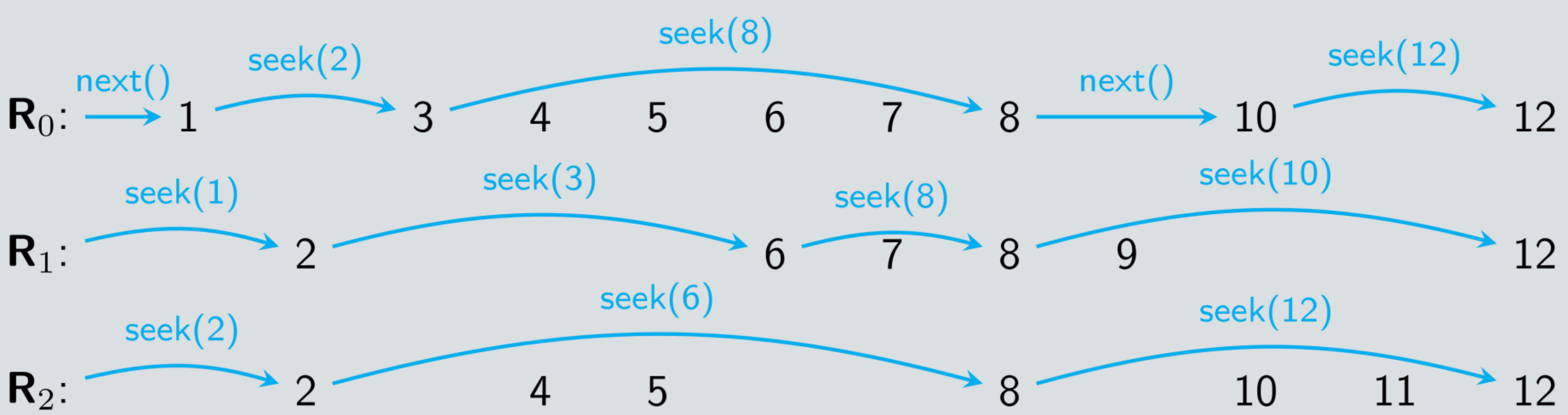
$\mathbf{R}_0$.next()
i := 1
while $\mathbf{R}_0$.key() **do**
  if $\mathbf{R}_i$.key() $==$ $\mathbf{R}_{(i-1) \bmod n}$.key() **then**
    return $\mathbf{R}_i$.key()
  else
    $\mathbf{R}_i$.seek($\mathbf{R}_{(i-1) \bmod n}$.key())
    $i := (i+1) \bmod n$

Unary joins

Leapfrog-next():

    $\mathbf{R}_0$.next()
    i := 1
    while $\mathbf{R}_0$.key() **do**
        if $\mathbf{R}_i$.key() $==$ $\mathbf{R}_{(i-1) \bmod n}$.key() **then**
            return $\mathbf{R}_i$.key()
        else
            $\mathbf{R}_i$.seek($\mathbf{R}_{(i-1) \bmod n}$.key())
            $i := (i+1) \bmod n$

- $\mathbf{R}_i$.begin() : get *before* the first value
- $\mathbf{R}_i$.key() : return the value at current position
- $\mathbf{R}_i$.next() : advance to the next position
- $\mathbf{R}_i$.seek(k) : advance to first element $\geq k$

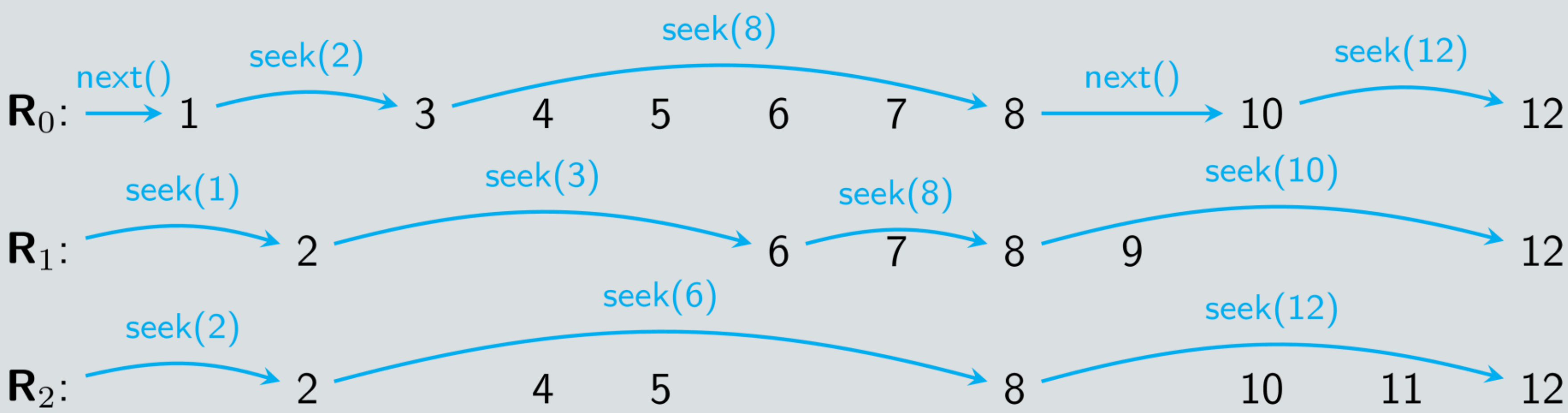


Runtime

Leapfrog-next():

$\mathbf{R}_0$.next()
i := 1
while $\mathbf{R}_0$.key() **do**
  if $\mathbf{R}_i.\text{key}() == \mathbf{R}_{(i-1) \bmod n}.\text{key}()$ **then**
    return $\mathbf{R}_i$.key()
  else
    $\mathbf{R}_i.\text{seek}(\mathbf{R}_{(i-1) \bmod n}.\text{key}())$
    $i := (i+1) \bmod n$

$$\mathcal{O}\left(n \cdot (\min_i |\mathbf{R}_i|) \cdot log(\max_i |\mathbf{R}_i|)\right)$$



Runtime

Leapfrog-next():

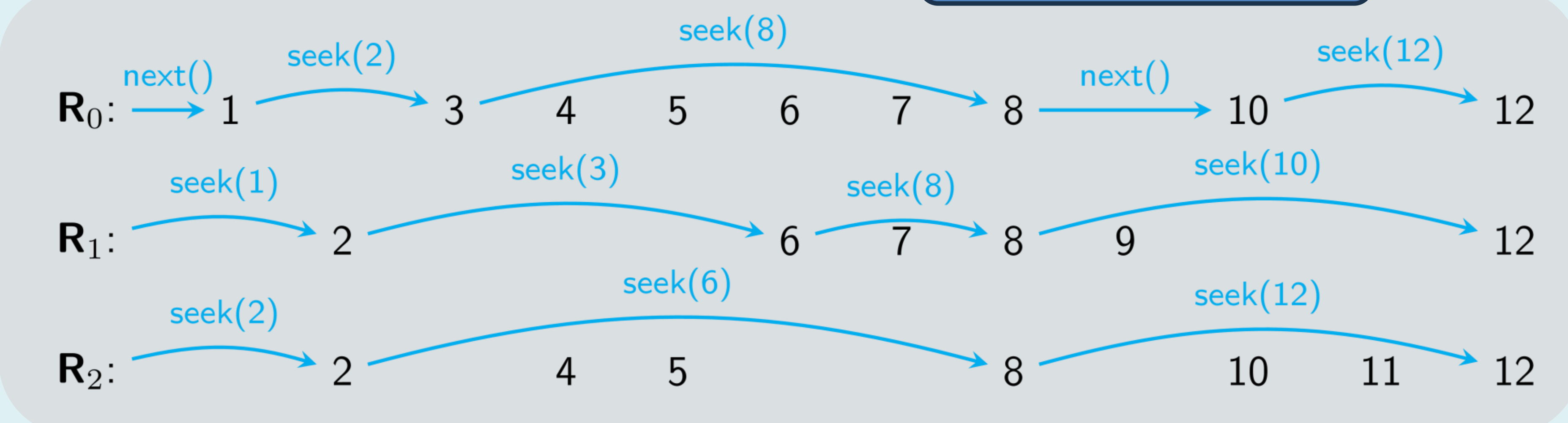
  $\mathbf{R}_0$.next()
  i := 1
  while $\mathbf{R}_0$.key() **do**
    if $\mathbf{R}_i$.key() $==$ $\mathbf{R}_{(i-1) \bmod n}$.key() **then**
      return $\mathbf{R}_i$.key()
    else
      $\mathbf{R}_i$.seek($\mathbf{R}_{(i-1) \bmod n}$.key())
      $i := (i+1) \bmod n$

Cost of a seek

Cycle through the iters

$$\mathcal{O}\Big(n \cdot (\min_i |\mathbf{R}_i|) \cdot log(\max_i |\mathbf{R}_i|)\Big)$$

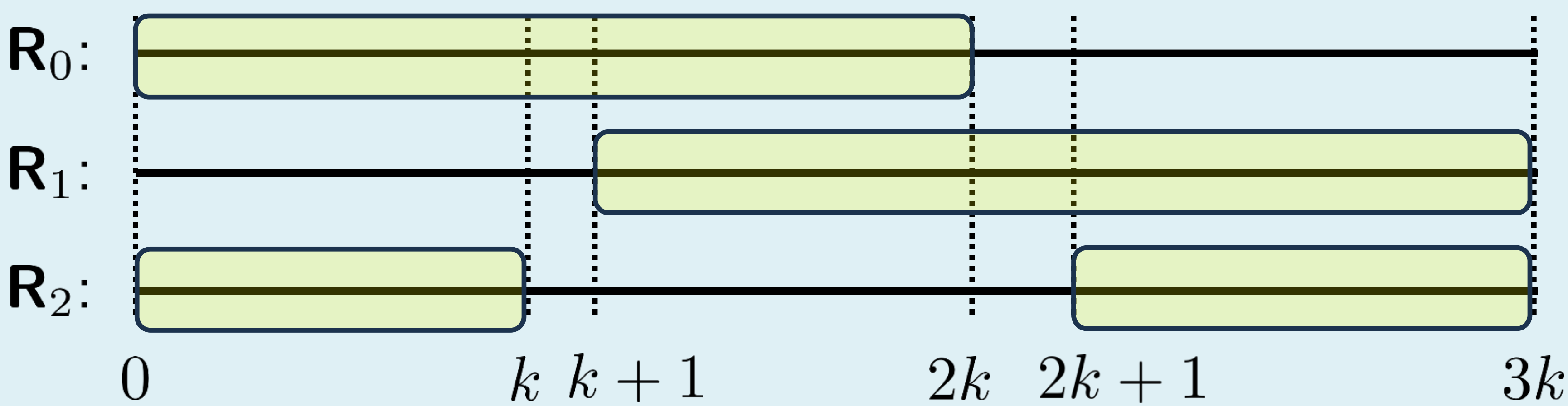
Max number of seeks



Runtime

```
Leapfrog-next():
    R_0.next()
    i := 1
    while R_0.key() do
        if R_i.key() == R_{(i-1) mod n}.key() then
            return R_i.key()
        else
            R_i.seek(R_{(i-1) mod n}.key())
            i := (i + 1) mod n
```

How many steps does the algorithm take to detect there are 0 results?



Runtime

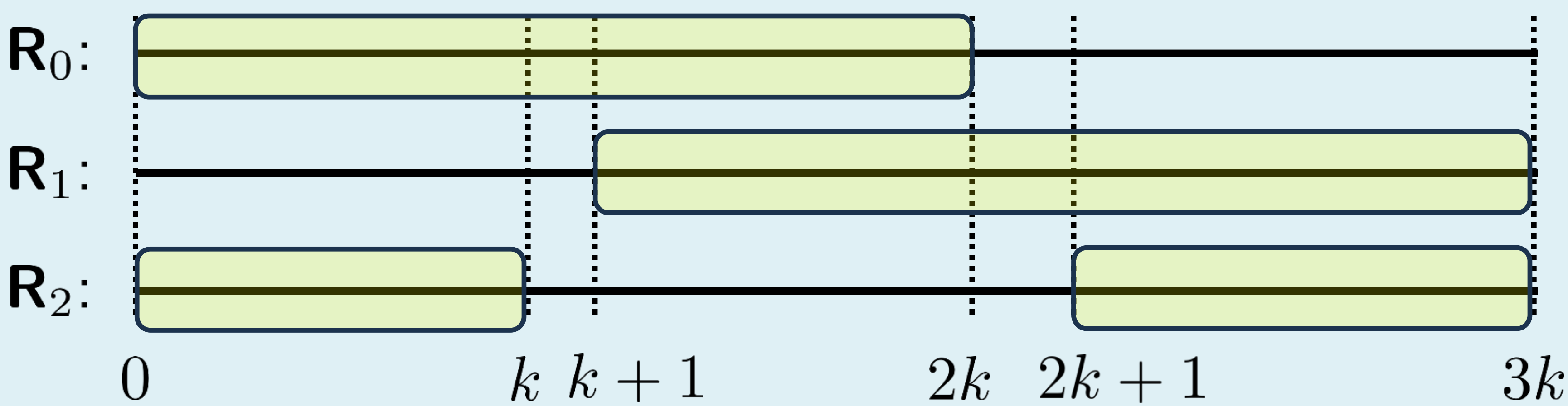
Leapfrog-next():

```
R_0.next()
i := 1
while R_0.key() do
    if R_i.key() == R_((i-1) mod n).key() then
        return R_i.key()
    else
        R_i.seek(R_((i-1) mod n).key())
        i := (i + 1) mod n
```

leapfrog: $\mathcal{O}(1)$

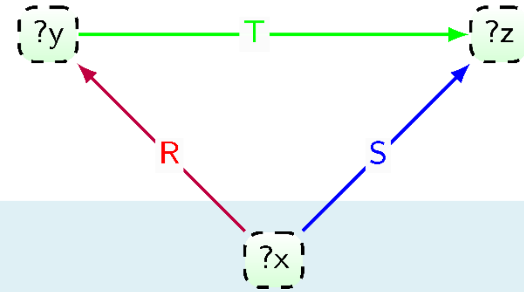
pairwise: $|\mathbf{R}_i \bowtie \mathbf{R}_j| = k$

How many steps does the algorithm take to detect there are 0 results?



Leapfrog Triejoin
(now with relations)

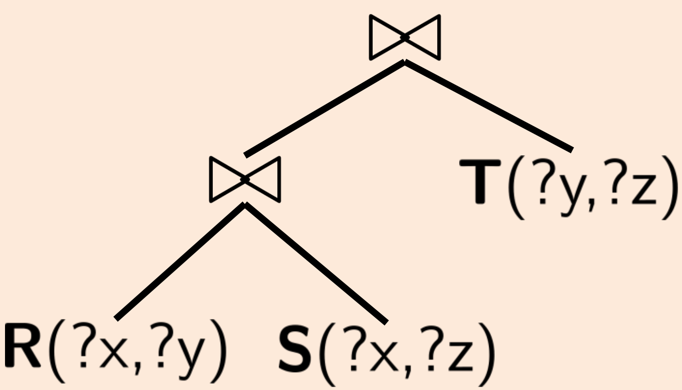
Evaluation of a join query



$$Q(?x,?y,?z) = \mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$$

Different evaluation philosophy

pairwise joins



Join at a time strategy

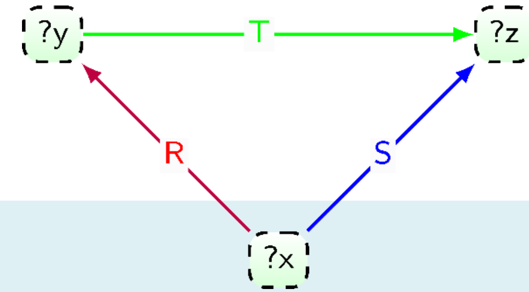
Leapfrog Triejoin

First ?x, then ?y, then ?z:

for each ?x that makes sense **do**
    for each ?y that makes sense **do**
        (Given this particular ?x)
        for each ?z that makes sense **do**
            (Given these particular ?x, ?y)
            return (?x,?y,?z)

Variable at a time strategy

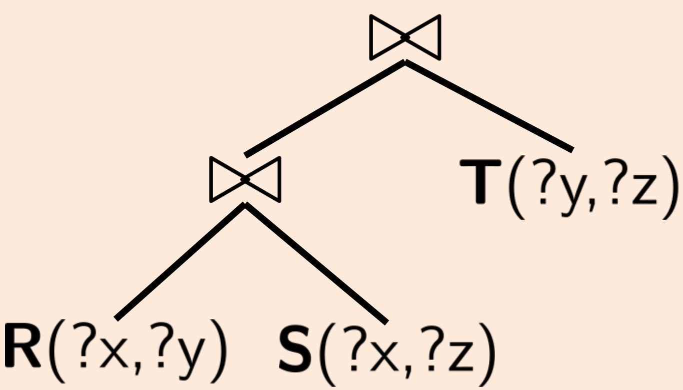
Evaluation of a join query



$$Q(?\mathrm{x},?\mathrm{y},?\mathrm{z}) = \mathbf{R}(?\mathrm{x},?\mathrm{y}) \bowtie \mathbf{S}(?\mathrm{x},?\mathrm{z}) \bowtie \mathbf{T}(?\mathrm{y},\ ?\mathrm{z})$$

Different evaluation philosophy

pairwise joins



Join at a time strategy

Leapfrog Triejoin

First ?x, then ?y, then ?z:

for each $a \in \mathbf{R}(?\mathrm{x}, \_) \cap \mathbf{S}(?\mathrm{x}, \_)$ **do**
  for each $b \in \mathbf{R}(a, ?y) \cap \mathbf{T}(?y, \_)$ **do**
    for each $c \in \mathbf{S}(a, ?z) \cap \mathbf{T}(b, ?z)$ **do**
      return (a, b, c)

Variable at a time strategy

Leapfrog Triejoin

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Global Variable Ordering (GAO)

Fix an order of query variables

Say $y_1, y_2, \ldots, y_m$

In each $\mathbf{R}_i(\overline{x_i})$

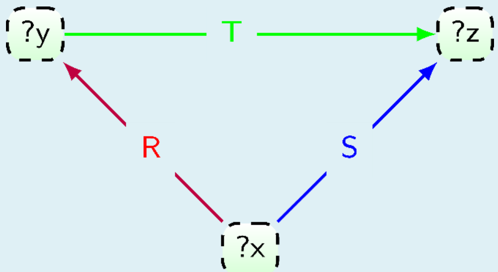
$\overline{x_i}$ are ordered

according to $y_1, \ldots, y_m$

Leapfrog Triejoin

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Global Variable Ordering (GAO)



?x,?y,?z $\Rightarrow$ **R**(?x,?y) $\bowtie$ **S**(?x,?z) $\bowtie$ **T**(?y, ?z)

?x,?z,?y $\Rightarrow$ **R**(?x,?y) $\bowtie$ **S**(?x,?z) $\bowtie$ **T**(?z, ?y)

?y,?z,?x $\Rightarrow$ **R**(?y,?x) $\bowtie$ **S**(?z,?x) $\bowtie$ **T**(?y, ?z)

Fix an order of query variables

Say $y_1, y_2, \ldots, y_m$

In each $\mathbf{R}_i(\overline{x_i})$

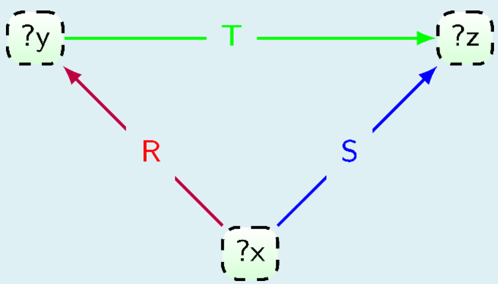
$\overline{x_i}$ are ordered

according to $y_1, \ldots, y_m$

Leapfrog Triejoin

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Global Variable Ordering (GAO)



$?x,?y,?z \;\Rightarrow\; \mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \boxed{\mathbf{T}(?y,\ ?z)}$

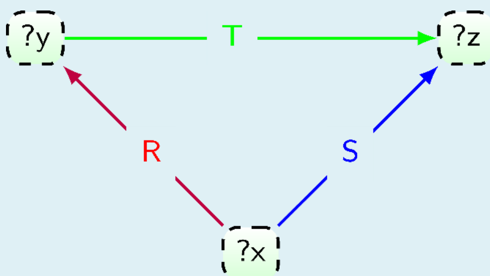
$?x,?z,?y \;\Rightarrow\; \mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \boxed{\mathbf{T}(?z,\ ?y)}$

$?y,?z,?x \;\Rightarrow\; \mathbf{R}(?y,?x) \bowtie \mathbf{S}(?z,?x) \bowtie \mathbf{T}(?y,\ ?z)$

Leapfrog Triejoin

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Global Variable Ordering (GAO)



?x,?y,?z $\Rightarrow$ **R**(?x,?y) $\bowtie$ **S**(?x,?z) $\bowtie$ $\boxed{\mathbf{T}(?y,\ ?z)}$

?x,?z,?y $\Rightarrow$ **R**(?x,?y) $\bowtie$ **S**(?x,?z) $\bowtie$ $\boxed{\mathbf{T}(?z,\ ?y)}$

?y,?z,?x $\Rightarrow$ **R**(?y,?x) $\bowtie$ **S**(?z,?x) $\bowtie$ **T**(?y, [illegible])

T

?y	?z
1	3
1	7
2	4
3	5

T

?z	?y
3	1
7	1
4	2
5	3

Leapfrog Triejoin

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Partially instantiating the join w.r.t. GAO

$$\mathbf{R}_i\left[a_1, \ldots, a_{n-1}, y_n\right] = \pi_{y_n}\left(\sigma_{y_1=a_1 \wedge y_2=a_2 \wedge \cdots \wedge y_{n-1}=a_{n-1}}(\mathbf{R}_i)\right)$$

Leapfrog Triejoin

$$Q(\overbrace{y_1, y_2, \ldots, y_m}^{\text{GAO}}) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Partially instantiating the join w.r.t. GAO

$$\mathbf{R}_i\left[\underbrace{a_1, \ldots, a_{n-1}}_{\text{some values}}, y_n\right] = \pi_{y_n}\left(\sigma_{\underbrace{y_1=a_1 \wedge y_2=a_2 \wedge \cdots \wedge y_{n-1}=a_{n-1}}_{\text{goes in the GAO order } (y_1 \text{ then } y_2 \cdots)}}(\mathbf{R}_i)\right)$$

Leapfrog Triejoin

$$Q(\overbrace{y_1, y_2, \ldots, y_m}^{\text{GAO}}) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Partially instantiating the join w.r.t. GAO

$$\mathbf{R}_i\left[\underbrace{a_1, \ldots, a_{n-1}}_{\text{some values}}, y_n\right] = \pi_{y_n}\left(\underbrace{\sigma_{y_1=a_1 \wedge y_2=a_2 \wedge \cdots \wedge y_{n-1}=a_{n-1}}}_{\text{ignore variables not in } \mathbf{R}_i}(\mathbf{R}_i)\right)$$

Leapfrog Triejoin

$$Q(\overbrace{y_1, y_2, \ldots, y_m}^{\text{GAO}}) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Partially instantiating the join w.r.t. GAO

$$\mathbf{R}_i\left[a_1, \ldots, a_{n-1}, y_n\right] = \pi_{y_n}\Big(\underbrace{\sigma_{y_1=a_1 \wedge y_2=a_2 \wedge \cdots \wedge y_{n-1}=a_{n-1}}}_{\text{ignore variables not in } \mathbf{R}_i}(\mathbf{R}_i)\Big)$$

x, y, z, w our GAO

T

x	y	w
1	3	2
1	3	5
1	3	9
1	4	6
3	5	7

$$\mathbf{T}\left[x\right] = \{1, 3\}$$

$$\mathbf{T}\left[1, y\right] = \{3, 4\}$$

$$\mathbf{T}\left[1, 3, 7, w\right] = \{2, 5, 9\}$$

ignore the z position

Leapfrog Triejoin

$$Q(\overbrace{y_1, y_2, \ldots, y_m}^{\text{GAO}}) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Partially instantiating the join w.r.t. GAO

$$\mathbf{R}_i\left[a_1, \ldots, a_{n-1}, y_n\right] = \pi_{y_n}\left(\underbrace{\sigma_{y_1=a_1 \wedge y_2=a_2 \wedge \cdots \wedge y_{n-1}=a_{n-1}}(\mathbf{R}_i)}_{\text{ignore variables not in } \mathbf{R}_i}\right)$$

Partially instantiating the query w.r.t. GAO

$$Q\left[a_1, \ldots, a_{n-1}, y_n\right] = \mathbf{R}_1\left[a_1, \ldots, a_{n-1}, y_n\right] \bowtie \cdots \bowtie \mathbf{R}_k\left[a_1, \ldots, a_{n-1}, y_n\right]$$

Leapfrog Triejoin

$$Q(\overbrace{y_1, y_2, \ldots, y_m}^{\text{GAO}}) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Partially instantiating the join w.r.t. GAO

$$\mathbf{R}_i\left[a_1, \ldots, a_{n-1}, y_n\right] = \pi_{y_n}\left(\underbrace{\sigma_{y_1=a_1 \wedge y_2=a_2 \wedge \cdots \wedge y_{n-1}=a_{n-1}}}_{\text{ignore variables not in } \mathbf{R}_i}(\mathbf{R}_i)\right)$$

Partially instantiating the query w.r.t. GAO

$$Q\left[a_1, \ldots, a_{n-1}, y_n\right] = \underbrace{\mathbf{R}_1\left[a_1, \ldots, a_{n-1}, y_n\right] \bowtie \cdots \bowtie \mathbf{R}_k\left[a_1, \ldots, a_{n-1}, y_n\right]}_{\text{ignore } \mathbf{R}_i \text{ if it does not contain } y_n}$$

Leapfrog Triejoin

$$Q(\overbrace{y_1, y_2, \ldots, y_m}^{\text{GAO}}) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

Partially instantiating the join w.r.t. GAO

$$\mathbf{R}_i\left[a_1, \ldots, a_{n-1}, y_n\right] = \pi_{y_n}\left(\sigma_{\underbrace{y_1=a_1 \wedge y_2=a_2 \wedge \cdots \wedge y_{n-1}=a_{n-1}}_{\text{ignore variables not in } \mathbf{R}_i}}(\mathbf{R}_i)\right)$$

Partially instantiating the query w.r.t. GAO

$$Q\left[a_1, \ldots, a_{n-1}, y_n\right] = \underbrace{\mathbf{R}_1\left[a_1, \ldots, a_{n-1}, y_n\right] \bowtie \cdots \bowtie \mathbf{R}_k\left[a_1, \ldots, a_{n-1}, y_n\right]}_{\text{unary join}}$$

Leapfrog Triejoin

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

$$Q\left[a_1, \ldots, a_{n-1}, y_n\right] = \mathbf{R}_1\left[a_1, \ldots, a_{n-1}, y_n\right] \bowtie \cdots \bowtie \mathbf{R}_k\left[a_1, \ldots, a_{n-1}, y_n\right]$$

Leapfrog-TrieJoin(GAO $= y_1, \ldots, y_m$):

- **for each** $a_1 \in$ Leapfrog-next($Q\left[y_1\right]$) **do**
  - **for each** $a_2 \in$ Leapfrog-next($Q\left[a_1, y_2\right]$) **do**
    - **for each** $a_3 \in$ Leapfrog-next($Q\left[a_1, a_2, y_3\right]$) **do**
      - $\ldots$
      - **for each** $a_m \in$ Leapfrog-next($Q\left[a_1, a_2, \ldots, a_{m-1}, y_m\right]$) **do**
        - Solutions $\leftarrow (a_1, a_2, \ldots, a_m)$

Leapfrog Triejoin

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie$$

$$Q\left[a_1, \ldots, a_{n-1}, y_n\right] = \mathbf{R}_1\left[a_1, \ldots, a_{n-1},\right.$$

Leapfrog-next():

  $\mathbf{R}_0$.next()

  i := 1

  while $\mathbf{R}_0$.key() **do**

    if $\mathbf{R}_i$.key() $==$ $\mathbf{R}_{(i-1) \bmod n}$.key() **then**

      return $\mathbf{R}_i$.key()

    else

      $\mathbf{R}_i$.seek($\mathbf{R}_{(i-1) \bmod n}$.key())

      $i := (i+1) \bmod n$

Leapfrog-TrieJoin(GAO $= y_1, \ldots, y_m$)

  for each $a_1 \in$ Leapfrog-next($Q\left[y_1\right]$) **do**

    for each $a_2 \in$ Leapfrog-next($Q\left[a_1, y_2\right]$) **do**

      for each $a_3 \in$ Leapfrog-next($Q\left[a_1, a_2, y_3\right]$) **do**

        $\ldots$

        for each $a_m \in$ Leapfrog-next($Q\left[a_1, a_2, \ldots, a_{m-1}, y_m\right]$) **do**

          Solutions $\leftarrow (a_1, a_2, \ldots, a_m)$

Leapfrog Triejoin

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2(\overline{x_2}) \bowtie \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

A bunch of nested fors is optimal?

$$\ldots{}_{n-1}, y_n\big] \bowtie \cdots \bowtie \mathbf{R}_k\big[a_1, \ldots, a_{n-1}, y_n\big]$$

Leapfrog-TrieJoin(GAO $= y_1, \ldots, y_m$):
  for each $a_1 \in$ Leapfrog-next($Q\big[y_1\big]$) **do**
    for each $a_2 \in$ Leapfrog-next($Q\big[a_1, y_2\big]$) **do**
      for each $a_3 \in$ Leapfrog-next($Q\big[a_1, a_2, y_3\big]$) **do**
        $\ldots$
        for each $a_m \in$ Leapfrog-next($Q\big[a_1, a_2, \ldots, a_{m-1}, y_m\big]$) **do**
          Solutions $\leftarrow (a_1, a_2, \ldots, a_m)$

Leapfrog Triejoin

AGM bound is tight:
There is a case where you saturate all these intersections!

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2 \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

A bunch of nested fors is optimal?

$$\ldots{}_{n-1}, y_n \cdots \bowtie \mathbf{R}_k\left[a_1, \ldots, a_{n-1}, y_n\right]$$

Leapfrog-TrieJoin(GAO $= y_1, \ldots, y_m$):
  for each $a_1 \in$ Leapfrog-next($Q\left[y_1\right]$) **do**
    for each $a_2 \in$ Leapfrog-next($Q\left[a_1, y_2\right]$) **do**
      for each $a_3 \in$ Leapfrog-next($Q\left[a_1, a_2, y_3\right]$) **do**
        $\ldots$
        for each $a_m \in$ Leapfrog-next($Q\left[a_1, a_2, \ldots, a_{m-1}, y_m\right]$) **do**
          Solutions $\leftarrow (a_1, a_2, \ldots, a_m)$

Leapfrog Triejoin

AGM bound is tight:

There is a case where you saturate all these intersections!

$$Q(y_1, y_2, \ldots, y_m) = \mathbf{R}_1(\overline{x_1}) \bowtie \mathbf{R}_2 \quad \cdots \bowtie \mathbf{R}_k(\overline{x_k})$$

A bunch of nested fors is optimal?

$$\ldots_{n-1}, y_n \quad \cdots \bowtie \mathbf{R}_k\left[a_1, \ldots, a_{n-1}, y_n\right]$$

Leapfrog-TrieJoin(GAO $= y_1, \ldots, y_m$):
- **for each** $a_1 \in$ Leapfrog-next($Q\left[y_1\right]$) **do**
  - **for each** $a_2 \in$ Leapfrog-next($Q\left[a_1, y_2\right]$) **do**
    - **for each** $a_3 \in$ Leapfrog-next($Q\left[a_1, a_2, y_3\right]$) **do**
      - $\ldots$
        - **for each** $a_m \in$ Leapfrog-next($Q\left[a_1, a_2, \ldots, a_{m-1}, y_m\right]$) **do**
          - Solutions $\leftarrow (a_1, a_2, \ldots, a_m)$

It's worst-case optimal!

Where are the Tries?

$$Q\left[a_1, \ldots, a_{n-1}, y_n\right] = \mathbf{R}_1\left[a_1, \ldots, a_{n-1}, y_n\right] \bowtie \cdots \bowtie \mathbf{R}_k\left[a_1, \ldots, a_{n-1}, y_n\right]$$

Leapfrog-TrieJoin(GAO $= y_1, \ldots, y_m$):
  for each $a_1 \in$ Leapfrog-next($Q[y_1]$) **do**
    for each $a_2 \in$ Leapfrog-next($Q[a_1, y_2]$) **do**
      for each $a_3 \in$ Leapfrog-next($Q[a_1, a_2, y_3]$) **do**
        $\cdots$
        for each $a_m \in$ Leapfrog-next($Q[a_1, a_2, \ldots, a_{m-1}, y_m]$) **do**
          Solutions $\leftarrow (a_1, a_2, \ldots, a_m)$

LeapFrog-next():
  $\mathbf{R}_0$.next()
  i := 1
  while $\mathbf{R}_0$.key() **do**
    if $\mathbf{R}_i$.key() $==$ $\mathbf{R}_{(i-1) \bmod n}$.key() **then**
      return $\mathbf{R}_i$.key()
    else
      $\mathbf{R}_i$.seek($\mathbf{R}_{(i-1) \bmod n}$.key())
      $i := (i+1) \bmod n$

Where are the Tries?

$$Q\left[a_1, \ldots, a_{n-1}, y_n\right] = \mathbf{R}_1\left[a_1, \ldots, a_{n-1}, y_n\right] \bowtie \cdots \bowtie \mathbf{R}_k\left[a_1, \ldots, a_{n-1}, y_n\right]$$

Leapfrog-TrieJoin(GAO $= y_1, \ldots, y_m$):
  for each $a_1 \in$ Leapfrog-next($Q\left[y_1\right]$) **do**
    for each $a_2 \in$ Leapfrog-next($Q\left[a_1, y_2\right]$) **do**
      for each $a_3 \in$ Leapfrog-next($Q\left[a_1, a_2, y_3\right]$) **do**
        $\ldots$
        for each $a_m \in$ Leapfrog-next($Q\left[a_1, a_2, \ldots, a_{m-1}, y_m\right]$) **do**
          Solutions $\leftarrow (a_1, a_2, \ldots, a_m)$

To compute $Q[a_1, \ldots, a_{n-1}, y_n]$:

Iterator interface for

$\mathbf{R}_i[a_1, \ldots, a_{n-1}, y_n]$

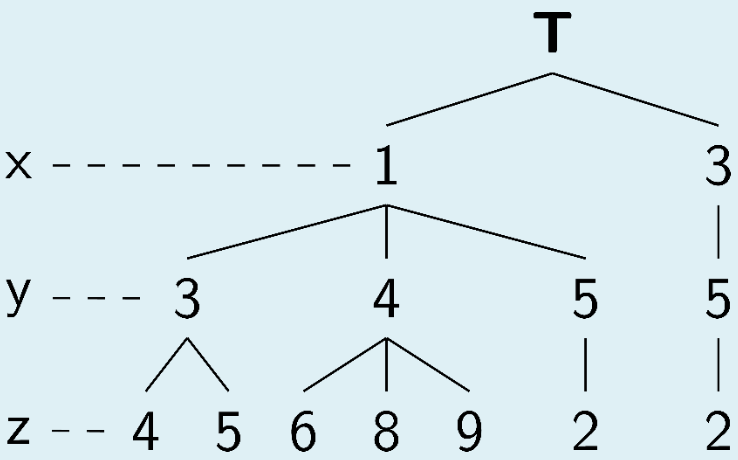
(with $O(log|\mathbf{R}_i|)$ seek)

LeapFrog-next():
  $\mathbf{R}_0$.next()
  i := 1
  while $\mathbf{R}_0$.key() **do**
    if $\mathbf{R}_i$.key() == $\mathbf{R}_{(i-1) \bmod n}$.key() **then**
      return $\mathbf{R}_i$.key()
    else
      $\mathbf{R}_i$.seek($\mathbf{R}_{(i-1) \bmod n}$.key())
      $i := (i+1) \bmod n$

Relation as a Trie

T

x	y	z
1	3	4
1	3	5
1	4	6
1	4	8
1	4	9
1	5	2
3	5	2

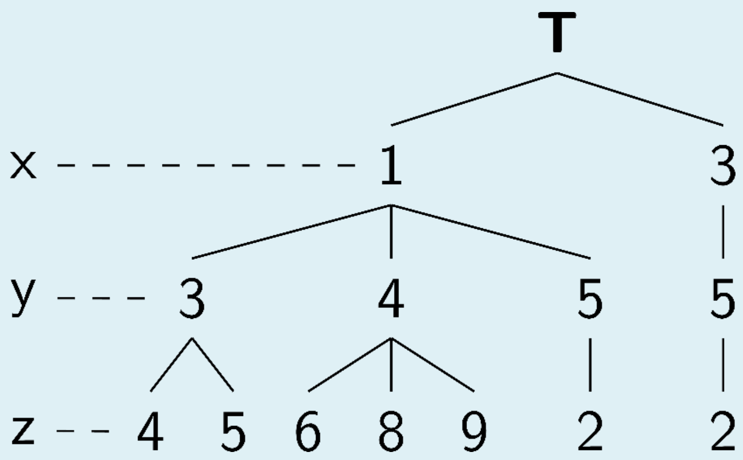


(Tikz image of the Trie by Cristian Riveros, example from [Leapfrog])

Relation as a Trie

T

x	y	z
1	3	4
1	3	5
1	4	6
1	4	8
1	4	9
1	5	2
3	5	2



Iterator interface

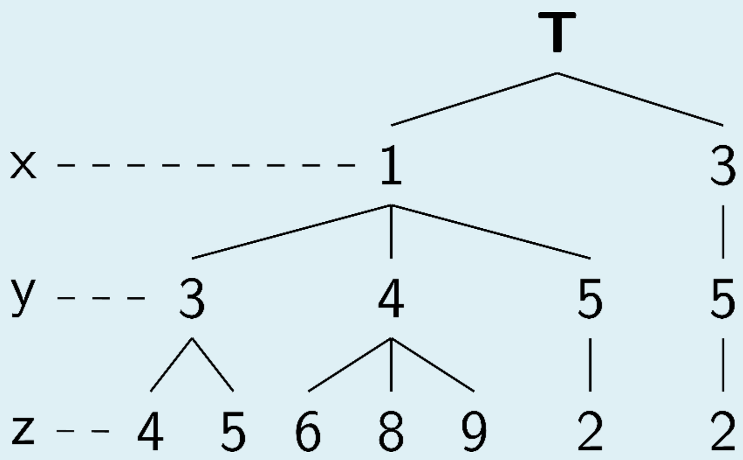
- $\mathbf{T}[a_1, \ldots a_{n-1}, y_n]$.begin() : get *before* the first value
- $\mathbf{T}[a_1, \ldots a_{n-1}, y_n]$.key() : return the value at current position
- $\mathbf{T}[a_1, \ldots a_{n-1}, y_n]$.next() : advance to the next position
- $\mathbf{T}[a_1, \ldots a_{n-1}, y_n]$.seek($k$) : advance to first element $\geq k$

(Tikz image of the Trie by Cristian Riveros, example from [Leapfrog])

Relation as a Trie

T

x	y	z
1	3	4
1	3	5
1	4	6
1	4	8
1	4	9
1	5	2
3	5	2



Iterator interface

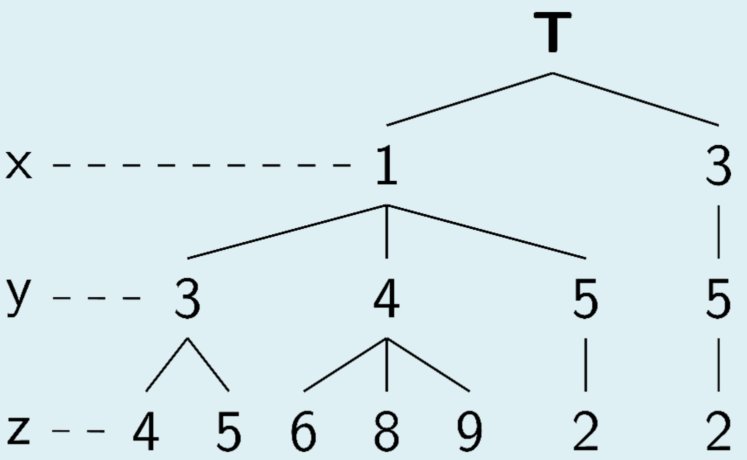
- $\mathbf{T}[1,4,y]$.begin() : before the leftmost child
- $\mathbf{T}[1,4,y]$.key() : value at current position
- $\mathbf{T}[1,4,y]$.next() : next sibling
- $\mathbf{T}[1,4,y]$.seek(k) : binary search $\mathcal{O}(log|\mathbf{T}|)$

(Tikz image of the Trie by Cristian Riveros, example from [Leapfrog])

Relations are usually Tries

T

x	y	z
1	3	4
1	3	5
1	4	6
1	4	8
1	4	9
1	5	2
3	5	2



Most common way to store a relation?

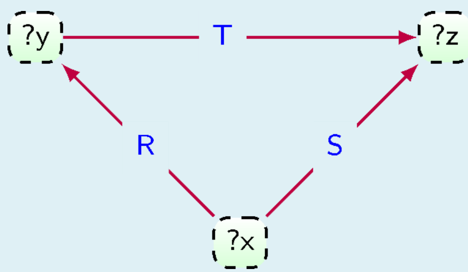
B+ tree

Supports search of a prefix of T[x,y,z] in O(log|T|)

Therefore seek can be done in the neccessary time

Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)

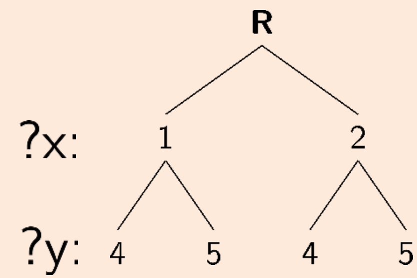


GAO ?x, ?y, ?z:

for each $x \in$ Leapfrog-next$(Q[?\text{x}])$ **do**
  for each $y \in$ Leapfrog-next$(Q[x, ?\text{y}])$ **do**
    for each $z \in$ Leapfrog-next$(Q[x, y, ?\text{z}])$ **do**
      Sol $\leftarrow (x, y, z)$

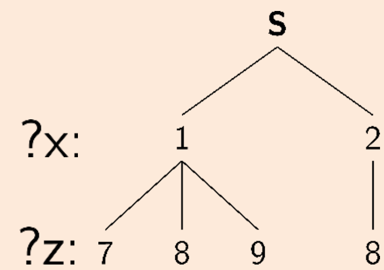
R

?x	?y
1	4
1	5
2	4
2	5



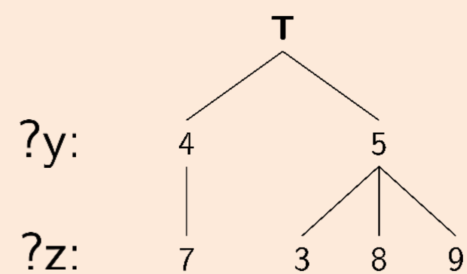
S

?x	?z
1	7
1	8
1	9
2	8



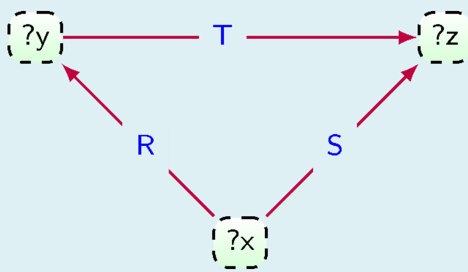
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$

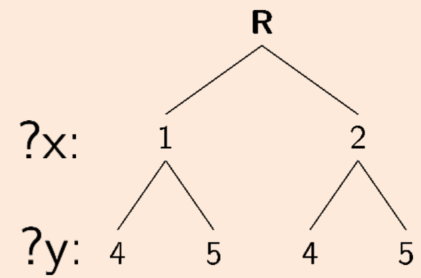


GAO ?x, ?y, ?z:

$\mathsf{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \mathsf{Valid}_{?x}$ **do**
  $\mathsf{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \mathsf{Valid}_{x,?y}$ **do**
    $\mathsf{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \mathsf{Valid}_{x,y?z}$ **do**
      $\mathsf{Sol} \leftarrow (x, y, z)$

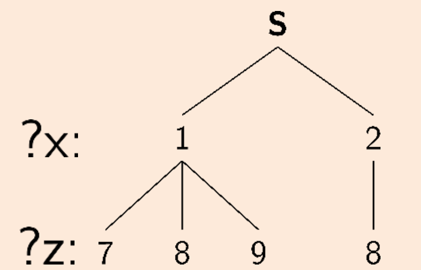
R

?x	?y
1	4
1	5
2	4
2	5



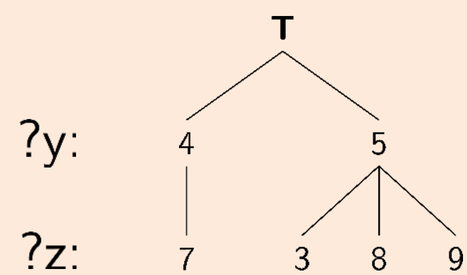
S

?x	?z
1	7
1	8
1	9
2	8



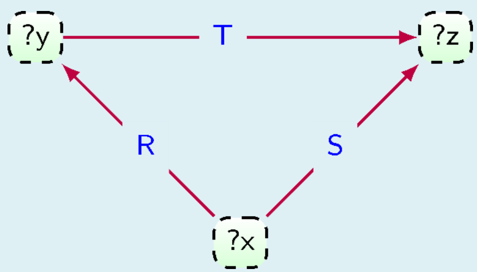
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)

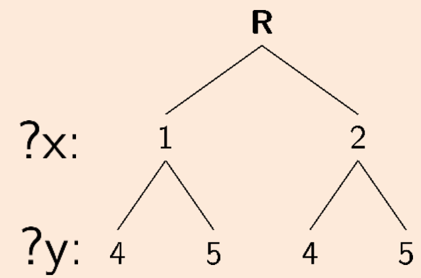


GAO ?x, ?y, ?z:

$\text{Valid}_{?\text{x}} \leftarrow \pi_{?\text{x}}(\mathbf{R}) \cap \pi_{?\text{x}}(\mathbf{S})$
for each $x \in \text{Valid}_{?\text{x}}$ **do**
  $\text{Valid}_{x,?\text{y}} \leftarrow \pi_{?\text{y}}(\mathbf{R}[x, ?\text{y}]) \cap \pi_{?\text{y}}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?\text{y}}$ **do**
    $\text{Valid}_{x,y,?\text{z}} \leftarrow \pi_{?\text{z}}(\mathbf{S}[x, ?\text{z}]) \cap \pi_{?\text{z}}(\mathbf{T}[y, ?\text{z}])$
    for each $z \in \text{Valid}_{x,y?\text{z}}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

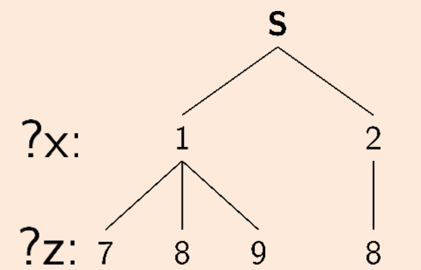
R

?x	?y
1	4
1	5
2	4
2	5



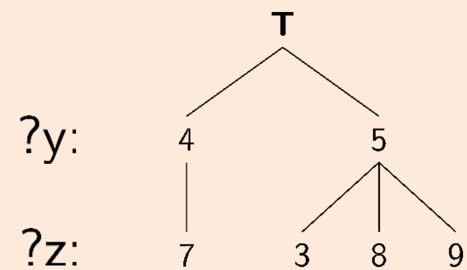
S

?x	?z
1	7
1	8
1	9
2	8



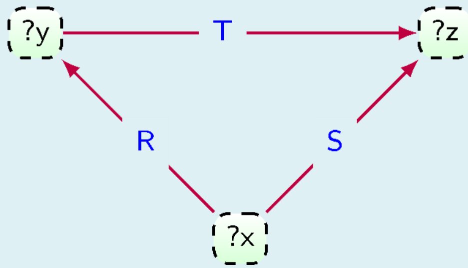
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)

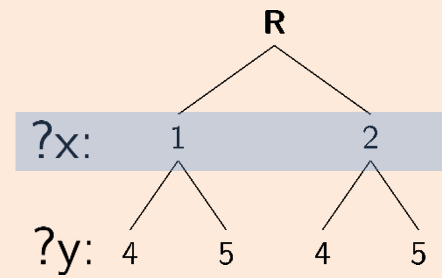


GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
    $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
    for each $y \in \text{Valid}_{x,?y}$ **do**
        $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
        for each $z \in \text{Valid}_{x,y?z}$ **do**
            $\text{Sol} \leftarrow (x, y, z)$

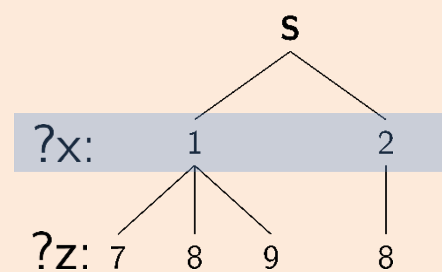
R

?x	?y
1	4
1	5
2	4
2	5



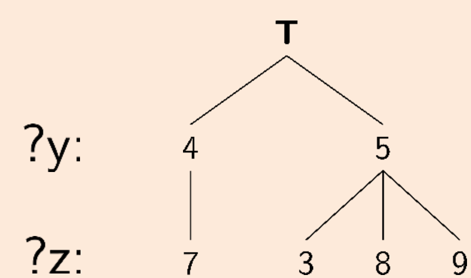
S

?x	?z
1	7
1	8
1	9
2	8



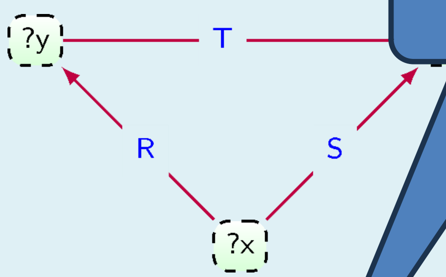
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



Unary leapfrog

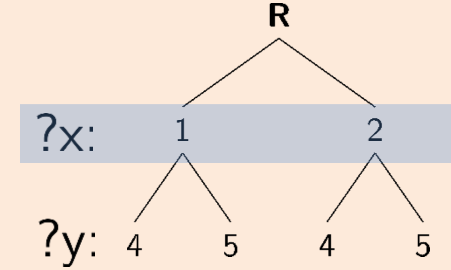
GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{1, 2\}$

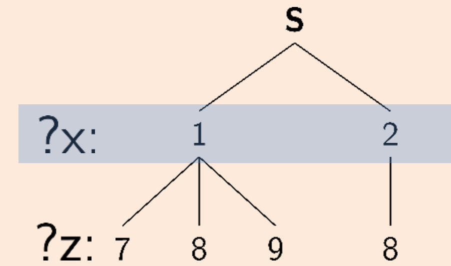
R

?x	?y
1	4
1	5
2	4
2	5



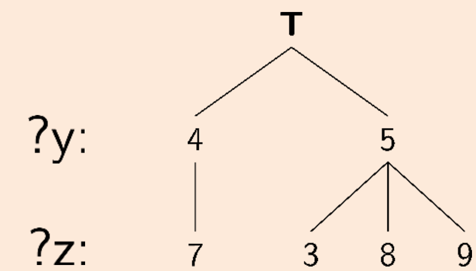
S

?x	?z
1	7
1	8
1	9
2	8



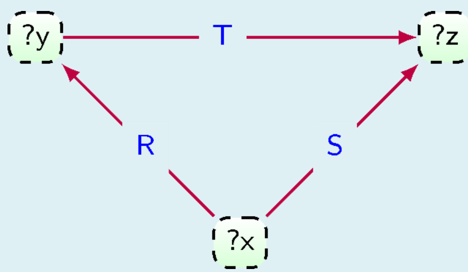
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,?z)$



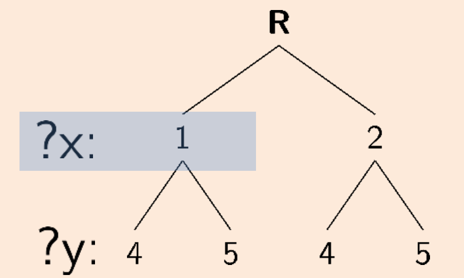
GAO ?x, ?y, ?z:

$\mathsf{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \mathsf{Valid}_{?x}$ **do**
  $\mathsf{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \mathsf{Valid}_{x,?y}$ **do**
    $\mathsf{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \mathsf{Valid}_{x,y?z}$ **do**
      $\mathsf{Sol} \leftarrow (x,y,z)$

$\mathsf{Valid}_{?x} = \{1,2\} \quad x = 1$

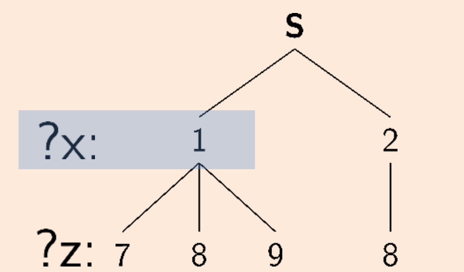
R

?x	?y
1	4
1	5
2	4
2	5



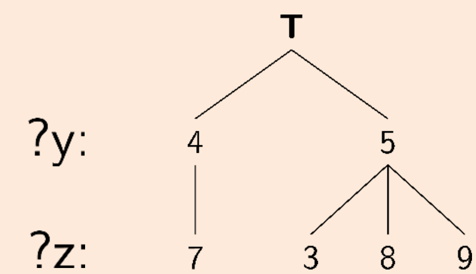
S

?x	?z
1	7
1	8
1	9
2	8



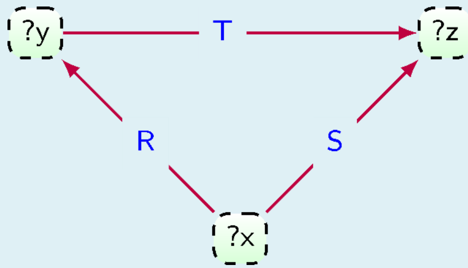
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) $\bowtie$ **S**(?x,?z) $\bowtie$ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{?\text{x}} \leftarrow \pi_{?\text{x}}(\mathbf{R}) \cap \pi_{?\text{x}}(\mathbf{S})$

for each $x \in \text{Valid}_{?\text{x}}$ **do**

    $\text{Valid}_{x,?\text{y}} \leftarrow \pi_{?\text{y}}(\mathbf{R}[x, ?\text{y}]) \cap \pi_{?\text{y}}(\mathbf{T})$

    for each $y \in \text{Valid}_{x,?\text{y}}$ **do**

        $\text{Valid}_{x,y,?\text{z}} \leftarrow \pi_{?\text{z}}(\mathbf{S}[x, ?\text{z}]) \cap \pi_{?\text{z}}(\mathbf{T}[y, ?\text{z}])$

        for each $z \in \text{Valid}_{x,y?\text{z}}$ **do**

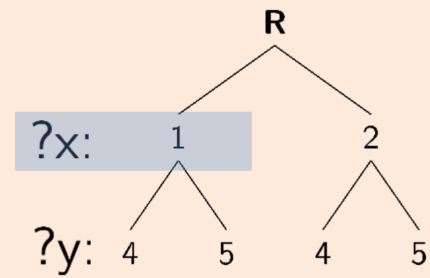
            $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?\text{x}} = \{1, 2\} \quad x = 1$

$\text{Valid}_{1,?\text{y}} = \{4, 5\}$

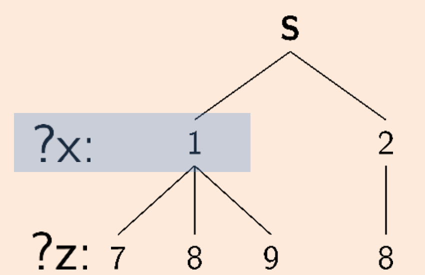
R

?x	?y
1	4
1	5
2	4
2	5



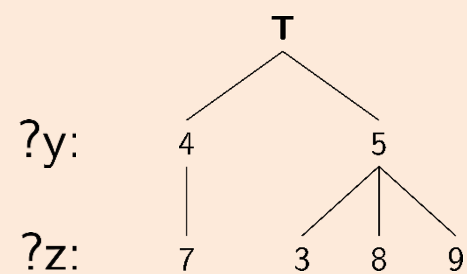
S

?x	?z
1	7
1	8
1	9
2	8



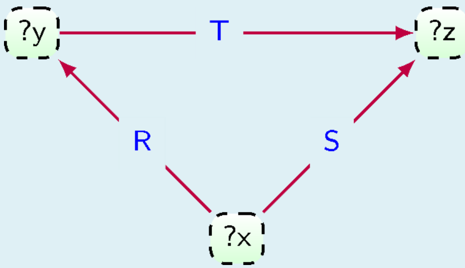
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,?z)$



GAO ?x, ?y, ?z:

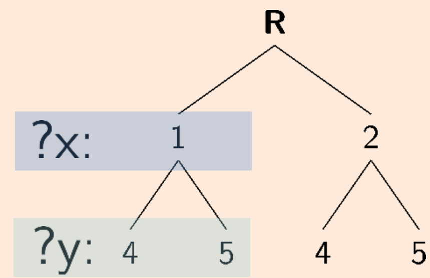
$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

$\text{Valid}_{?x} = \{1,2\} \quad x = 1$

$\text{Valid}_{1,?y} = \{4,5\}$

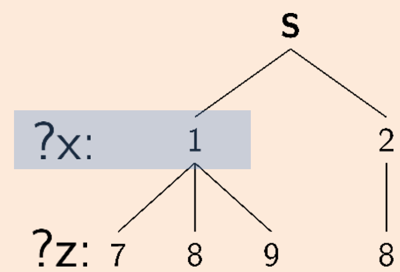
R

?x	?y
1	4
1	5
2	4
2	5



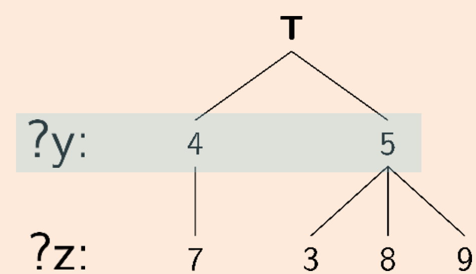
S

?x	?z
1	7
1	8
1	9
2	8



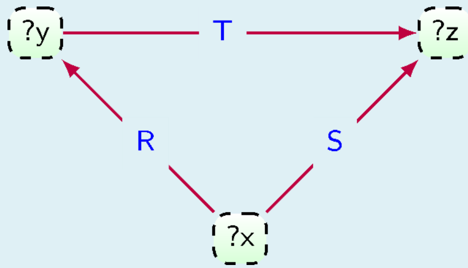
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

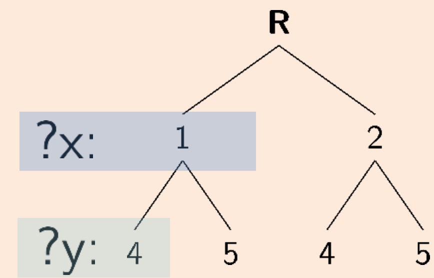
$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
    $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
    for each $y \in \text{Valid}_{x,?y}$ **do**
        $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
        for each $z \in \text{Valid}_{x,y?z}$ **do**
            $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{1, 2\} \quad x = 1$

$\text{Valid}_{1,?y} = \{4, 5\} \quad y = 4$

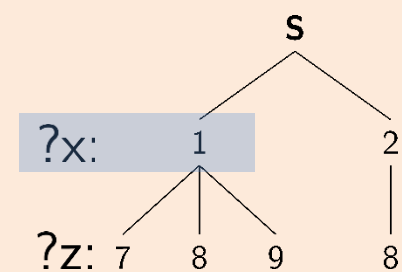
R

?x	?y
1	4
1	5
2	4
2	5



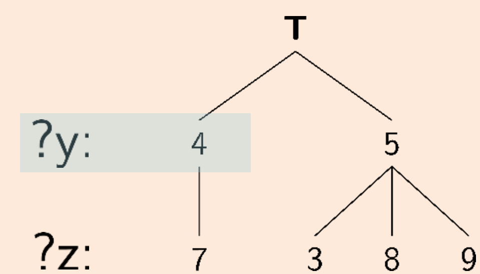
S

?x	?z
1	7
1	8
1	9
2	8



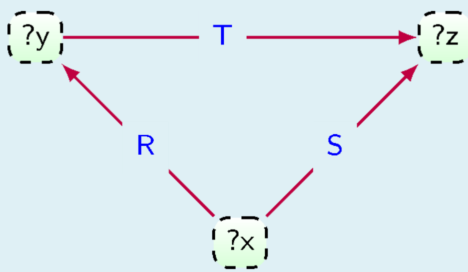
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{\text{?x}} \leftarrow \pi_{\text{?x}}(\mathbf{R}) \cap \pi_{\text{?x}}(\mathbf{S})$
for each $x \in \text{Valid}_{\text{?x}}$ **do**
    $\text{Valid}_{x,\text{?y}} \leftarrow \pi_{\text{?y}}(\mathbf{R}[x, \text{?y}]) \cap \pi_{\text{?y}}(\mathbf{T})$
    for each $y \in \text{Valid}_{x,\text{?y}}$ **do**
        $\text{Valid}_{x,y,\text{?z}} \leftarrow \pi_{\text{?z}}(\mathbf{S}[x, \text{?z}]) \cap \pi_{\text{?z}}(\mathbf{T}[y, \text{?z}])$
        for each $z \in \text{Valid}_{x,y\text{?z}}$ **do**
            $\text{Sol} \leftarrow (x, y, z)$

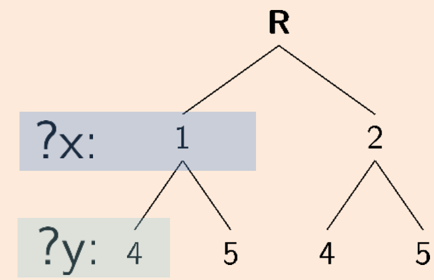
$\text{Valid}_{\text{?x}} = \{1, 2\} \quad x = 1$

$\text{Valid}_{1,\text{?y}} = \{4, 5\} \quad y = 4$

$\text{Valid}_{1,4,\text{?z}} = \{7\}$

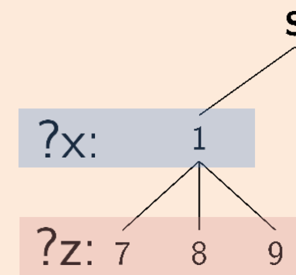
R

?x	?y
1	4
1	5
2	4
2	5



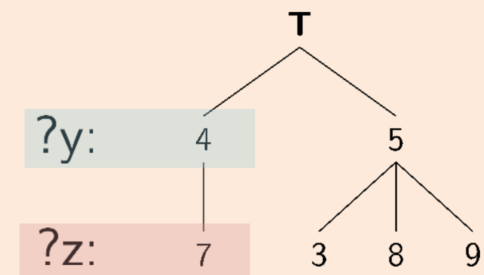
S

?x	?z
1	7
1	8
1	9
2	8



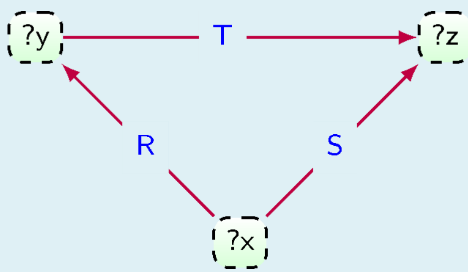
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

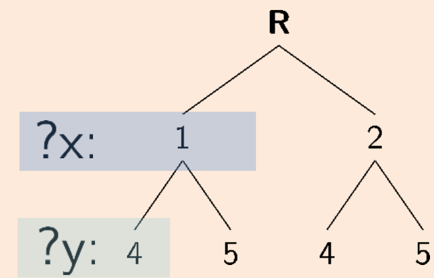
$\text{Valid}_{?x} = \{1, 2\} \quad x = 1$

$\text{Valid}_{1,?y} = \{4, 5\} \quad y = 4$

$\text{Valid}_{1,4,?z} = \{7\} \quad z = 7$

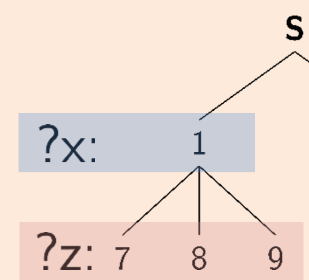
R

?x	?y
1	4
1	5
2	4
2	5



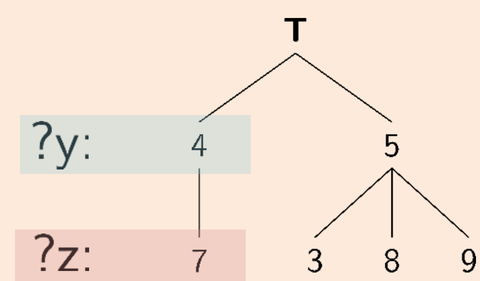
S

?x	?z
1	7
1	8
1	9
2	8



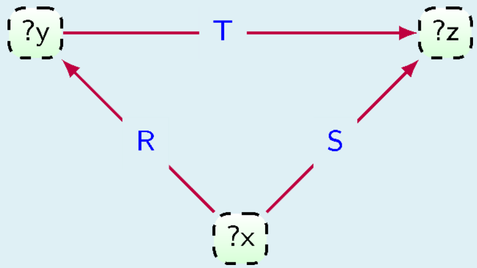
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

$\text{Valid}_{?x} = \{1,2\} \quad x = 1$

$\text{Valid}_{1,?y} = \{4,5\} \quad y = 4$

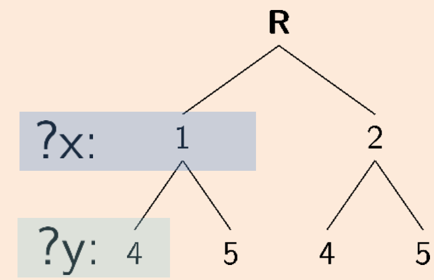
$\text{Valid}_{1,4,?z} = \{7\} \quad z = 7$

Sol

?x	?y	?z
1	4	7

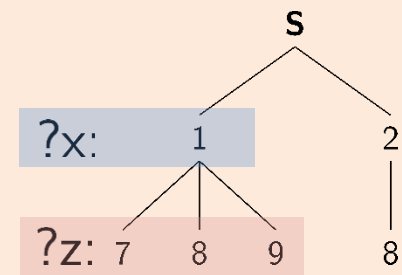
R

?x	?y
1	4
1	5
2	4
2	5



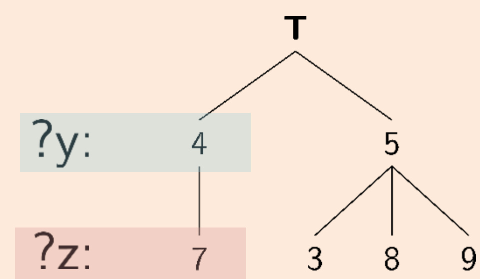
S

?x	?z
1	7
1	8
1	9
2	8



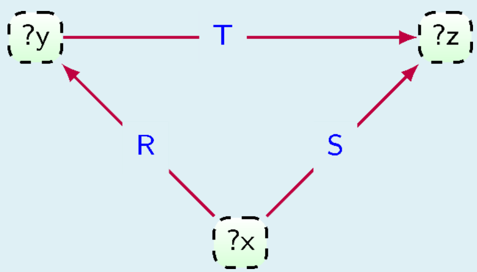
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\mathsf{Valid}_{?\mathsf{x}} \leftarrow \pi_{?\mathsf{x}}(\mathbf{R}) \cap \pi_{?\mathsf{x}}(\mathbf{S})$
for each $x \in \mathsf{Valid}_{?\mathsf{x}}$ **do**
  $\mathsf{Valid}_{x,?\mathsf{y}} \leftarrow \pi_{?\mathsf{y}}(\mathbf{R}[x, ?\mathsf{y}]) \cap \pi_{?\mathsf{y}}(\mathbf{T})$
  for each $y \in \mathsf{Valid}_{x,?\mathsf{y}}$ **do**
    $\mathsf{Valid}_{x,y,?\mathsf{z}} \leftarrow \pi_{?\mathsf{z}}(\mathbf{S}[x, ?\mathsf{z}]) \cap \pi_{?\mathsf{z}}(\mathbf{T}[y, ?\mathsf{z}])$
    for each $z \in \mathsf{Valid}_{x,y?\mathsf{z}}$ **do**
      $\mathsf{Sol} \leftarrow (x, y, z)$

$\mathsf{Valid}_{?\mathsf{x}} = \{1, 2\} \quad x = 1$

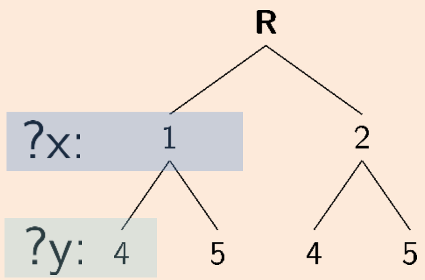
$\mathsf{Valid}_{1,?\mathsf{y}} = \{4, 5\} \quad y = 4$

Sol

?x	?y	?z
1	4	7

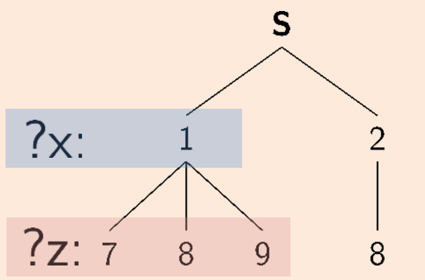
R

?x	?y
1	4
1	5
2	4
2	5



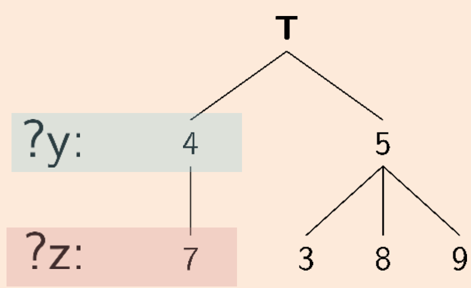
S

?x	?z
1	7
1	8
1	9
2	8



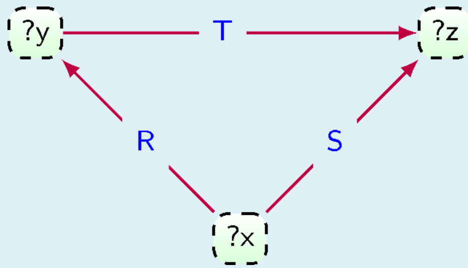
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

$\text{Valid}_{?x} = \{1,2\} \quad x = 1$

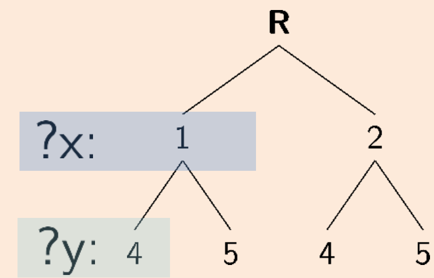
$\text{Valid}_{1,?y} = \{4,5\} \quad y = 4$

Sol

?x	?y	?z
1	4	7

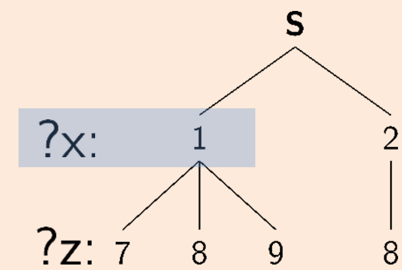
R

?x	?y
1	4
1	5
2	4
2	5



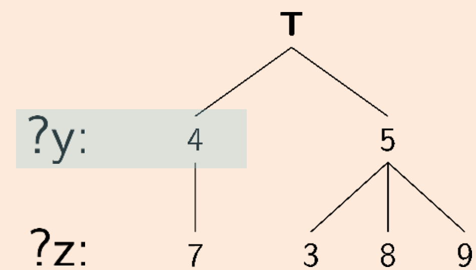
S

?x	?z
1	7
1	8
1	9
2	8



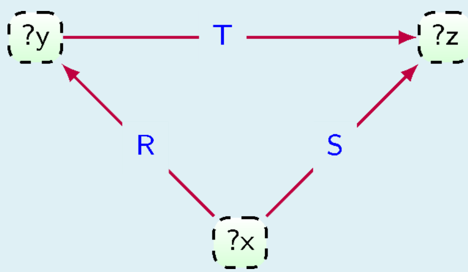
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
 $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
 for each $y \in \text{Valid}_{x,?y}$ **do**
  $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
  for each $z \in \text{Valid}_{x,y?z}$ **do**
   $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{1, 2\} \quad x = 1$

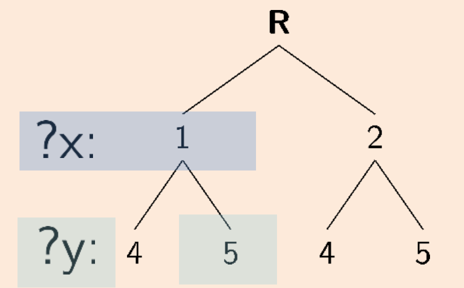
$\text{Valid}_{1,?y} = \{4, 5\} \quad y = 5$

Sol

?x	?y	?z
1	4	7

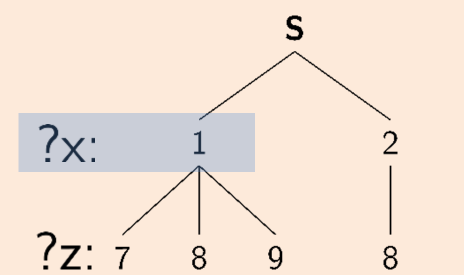
R

?x	?y
1	4
1	5
2	4
2	5



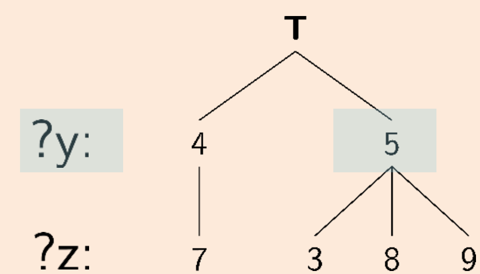
S

?x	?z
1	7
1	8
1	9
2	8



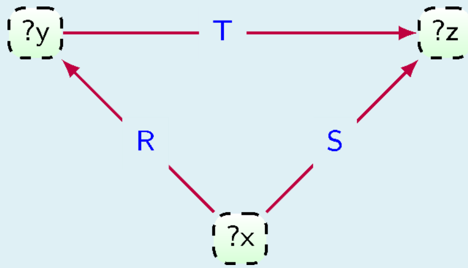
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{1, 2\} \quad x = 1$

$\text{Valid}_{1,?y} = \{4, 5\} \quad y = 5$

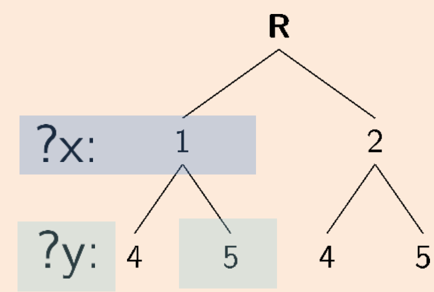
$\text{Valid}_{1,5,?z} = \{8, 9\}$

Sol

?x	?y	?z
1	4	7

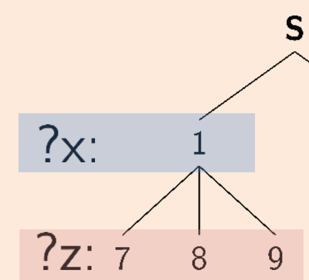
R

?x	?y
1	4
1	5
2	4
2	5



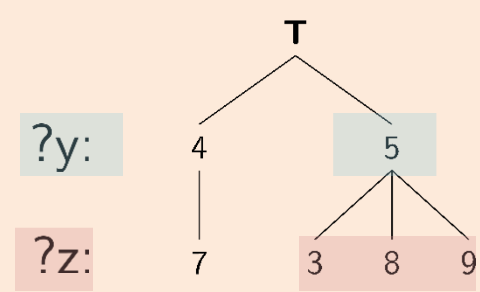
S

?x	?z
1	7
1	8
1	9
2	8



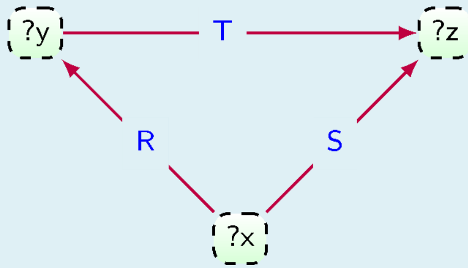
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$

for each $x \in \text{Valid}_{?x}$ **do**

$\quad \text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$

$\quad$ **for each** $y \in \text{Valid}_{x,?y}$ **do**

$\quad\quad \text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$

$\quad\quad$ **for each** $z \in \text{Valid}_{x,y?z}$ **do**

$\quad\quad\quad \text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{1, 2\} \quad x = 1$

$\text{Valid}_{1,?y} = \{4, 5\} \quad y = 5$

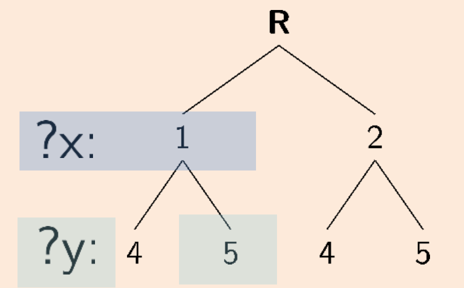
$\text{Valid}_{1,5,?z} = \{8, 9\}$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

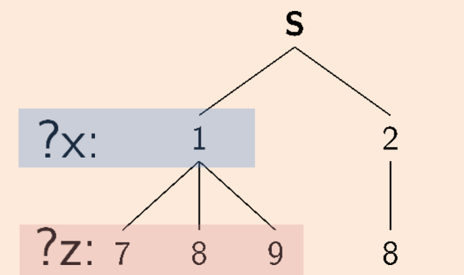
R

?x	?y
1	4
1	5
2	4
2	5



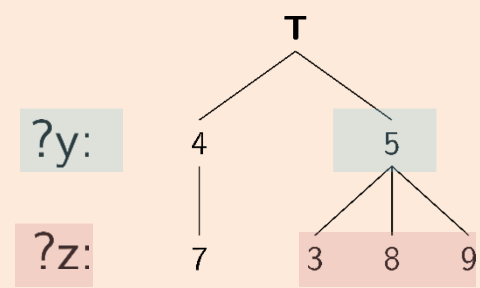
S

?x	?z
1	7
1	8
1	9
2	8



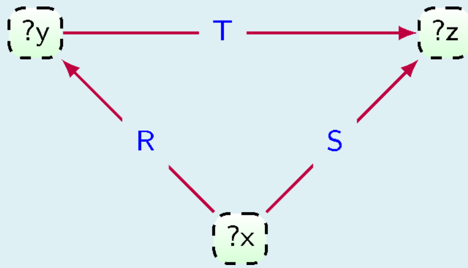
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{?\text{x}} \leftarrow \pi_{?\text{x}}(\mathbf{R}) \cap \pi_{?\text{x}}(\mathbf{S})$
for each $x \in \text{Valid}_{?\text{x}}$ **do**
$\quad \text{Valid}_{x,?\text{y}} \leftarrow \pi_{?\text{y}}(\mathbf{R}[x,?\text{y}]) \cap \pi_{?\text{y}}(\mathbf{T})$
$\quad$ **for each** $y \in \text{Valid}_{x,?\text{y}}$ **do**
$\quad\quad \text{Valid}_{x,y,?\text{z}} \leftarrow \pi_{?\text{z}}(\mathbf{S}[x,?\text{z}]) \cap \pi_{?\text{z}}(\mathbf{T}[y,?\text{z}])$
$\quad\quad$ **for each** $z \in \text{Valid}_{x,y?\text{z}}$ **do**
$\quad\quad\quad \text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?\text{x}} = \{1, 2\} \quad x = 1$

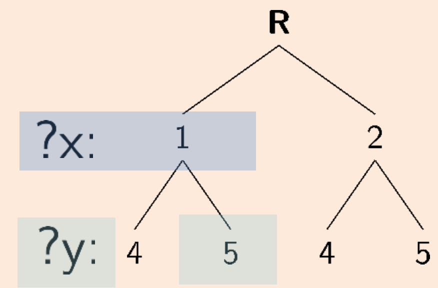
$\text{Valid}_{1,?\text{y}} = \{4, 5\} \quad y = 5$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

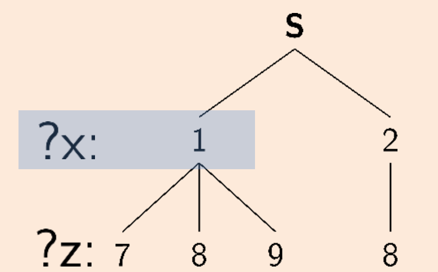
R

?x	?y
1	4
1	5
2	4
2	5



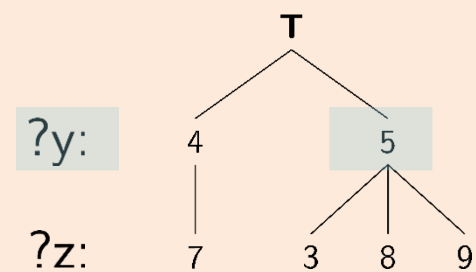
S

?x	?z
1	7
1	8
1	9
2	8



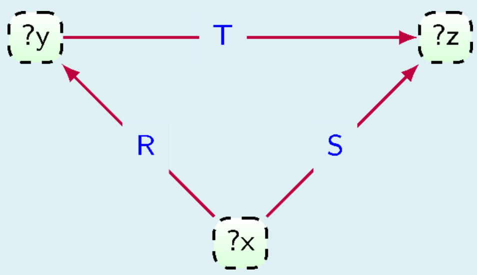
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\mathsf{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \mathsf{Valid}_{?x}$ **do**
  $\mathsf{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \mathsf{Valid}_{x,?y}$ **do**
    $\mathsf{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \mathsf{Valid}_{x,y?z}$ **do**
      $\mathsf{Sol} \leftarrow (x, y, z)$

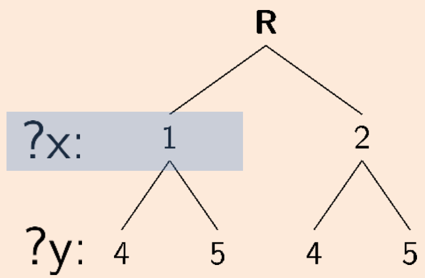
$\mathsf{Valid}_{?x} = \{1, 2\}$ $x = 1$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

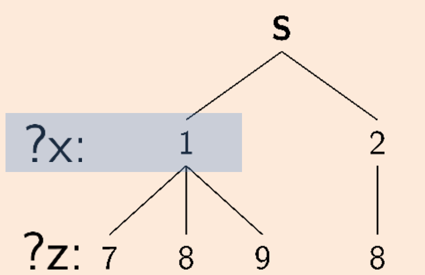
R

?x	?y
1	4
1	5
2	4
2	5



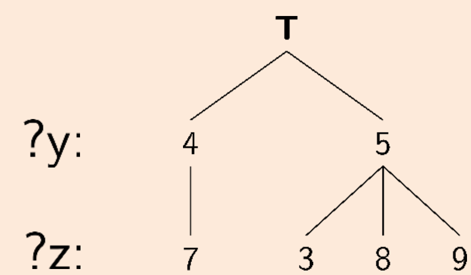
S

?x	?z
1	7
1	8
1	9
2	8



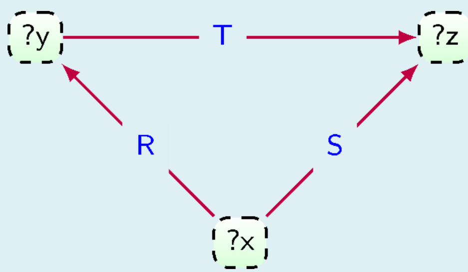
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$

for each $x \in \text{Valid}_{?x}$ **do**

  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$

  for each $y \in \text{Valid}_{x,?y}$ **do**

    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$

    for each $z \in \text{Valid}_{x,y?z}$ **do**

      $\text{Sol} \leftarrow (x, y, z)$

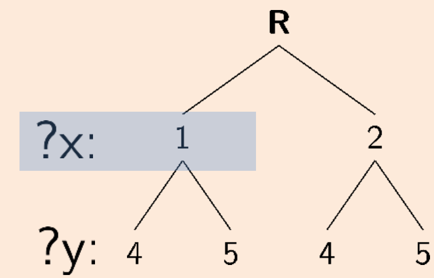
$\text{Valid}_{?x} = \{1, 2\}$ $x = 1$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

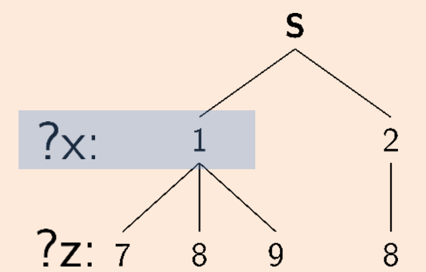
R

?x	?y
1	4
1	5
2	4
2	5



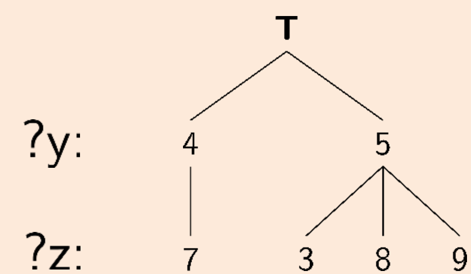
S

?x	?z
1	7
1	8
1	9
2	8



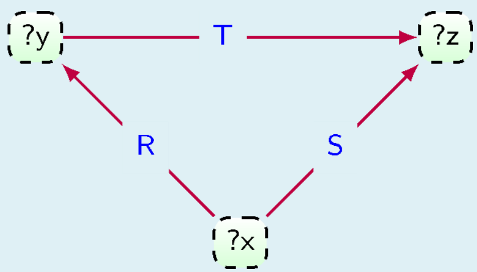
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

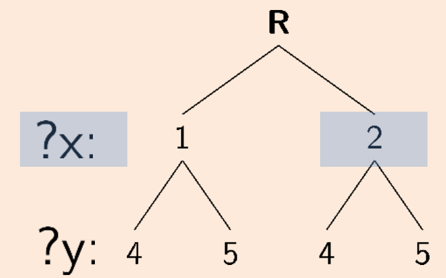
$\text{Valid}_{?x} = \{1,2\} \quad x = 2$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

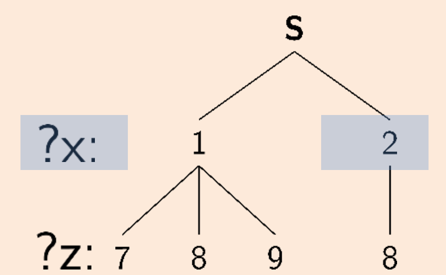
R

?x	?y
1	4
1	5
2	4
2	5



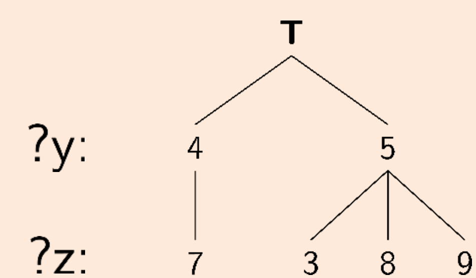
S

?x	?z
1	7
1	8
1	9
2	8



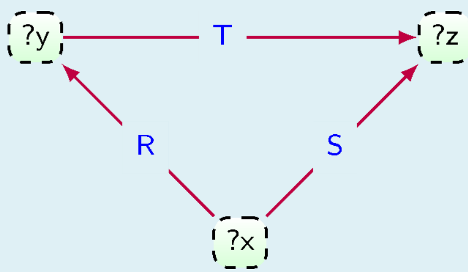
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{1, 2\} \quad x = 2$

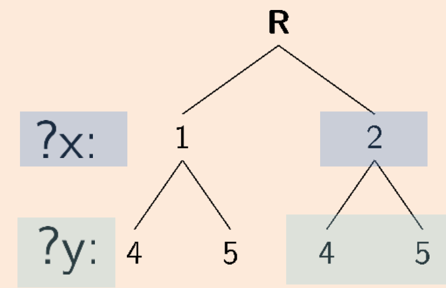
$\text{Valid}_{2,?y} = \{4, 5\}$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

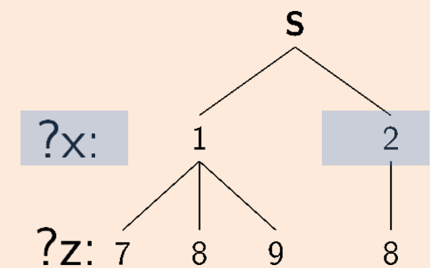
R

?x	?y
1	4
1	5
2	4
2	5



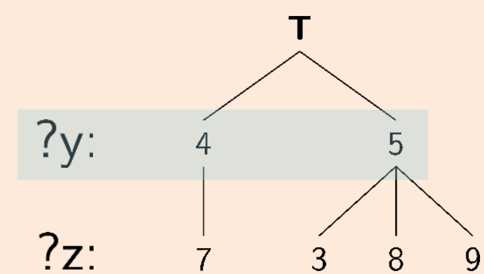
S

?x	?z
1	7
1	8
1	9
2	8



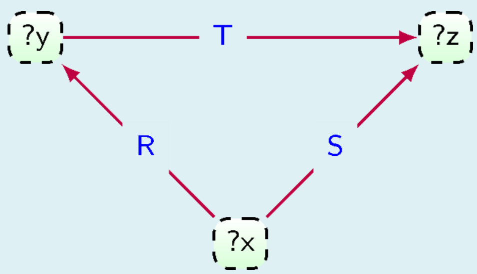
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{\text{?x}} \leftarrow \pi_{\text{?x}}(\mathbf{R}) \cap \pi_{\text{?x}}(\mathbf{S})$
for each $x \in \text{Valid}_{\text{?x}}$ **do**
  $\text{Valid}_{x,\text{?y}} \leftarrow \pi_{\text{?y}}(\mathbf{R}[x, \text{?y}]) \cap \pi_{\text{?y}}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,\text{?y}}$ **do**
    $\text{Valid}_{x,y,\text{?z}} \leftarrow \pi_{\text{?z}}(\mathbf{S}[x, \text{?z}]) \cap \pi_{\text{?z}}(\mathbf{T}[y, \text{?z}])$
    for each $z \in \text{Valid}_{x,y\text{?z}}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{\text{?x}} = \{1, 2\} \quad x = 2$

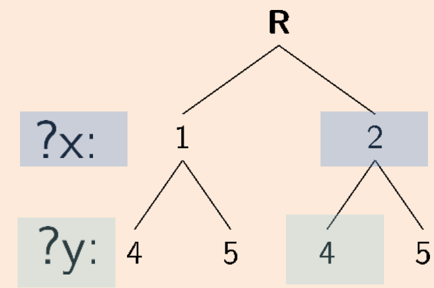
$\text{Valid}_{2,\text{?y}} = \{4, 5\} \quad y = 4$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

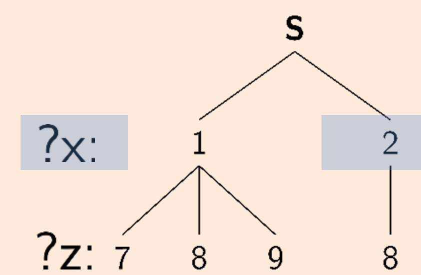
R

?x	?y
1	4
1	5
2	4
2	5



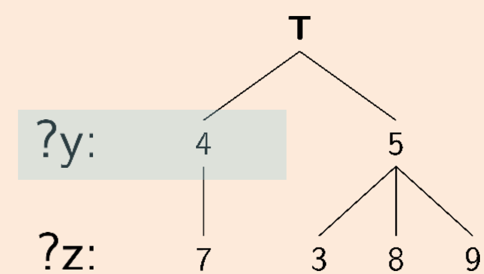
S

?x	?z
1	7
1	8
1	9
2	8



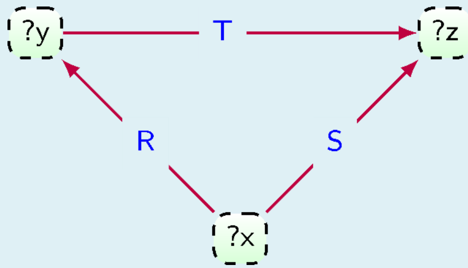
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

$\text{Valid}_{?x} = \{1,2\}$ $x = 2$

$\text{Valid}_{2,?y} = \{4,5\}$ $y = 4$

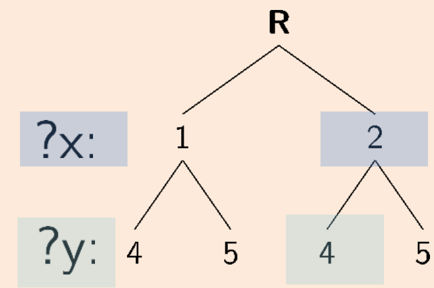
$\text{Valid}_{2,4,?z} = \emptyset$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

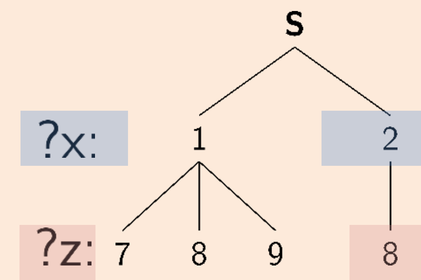
R

?x	?y
1	4
1	5
2	4
2	5



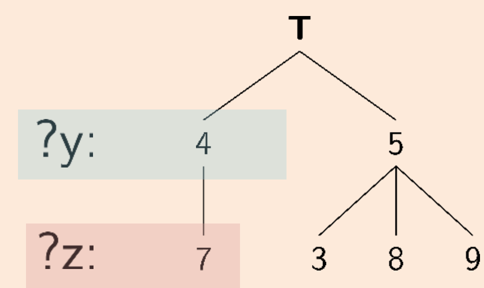
S

?x	?z
1	7
1	8
1	9
2	8



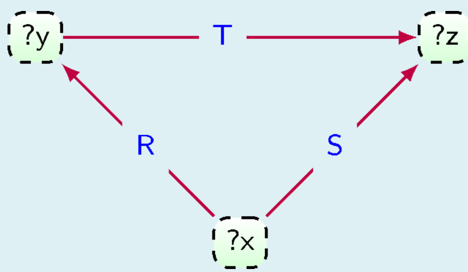
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
    $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
    for each $y \in \text{Valid}_{x,?y}$ **do**
        $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
        for each $z \in \text{Valid}_{x,y?z}$ **do**
            $\text{Sol} \leftarrow (x,y,z)$

$\text{Valid}_{?x} = \{1,2\} \quad x = 2$

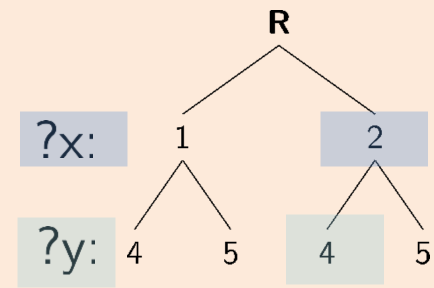
$\text{Valid}_{2,?y} = \{4,5\} \quad y = 4$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

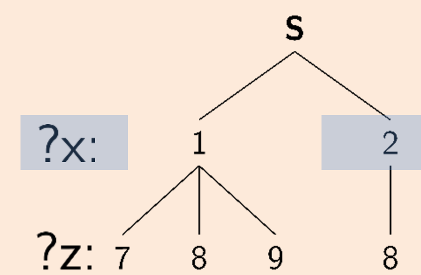
R

?x	?y
1	4
1	5
2	4
2	5



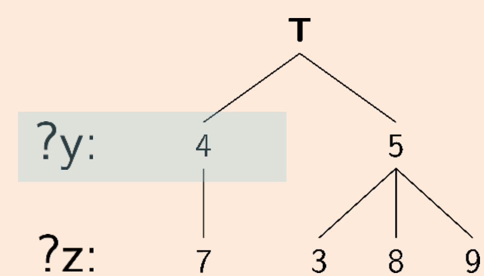
S

?x	?z
1	7
1	8
1	9
2	8



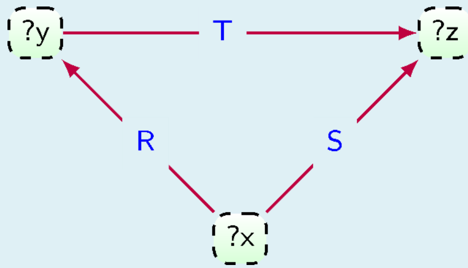
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$

for each $x \in \text{Valid}_{?x}$ **do**

  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$

  for each $y \in \text{Valid}_{x,?y}$ **do**

    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$

    for each $z \in \text{Valid}_{x,y?z}$ **do**

      $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{1, 2\}$ $x = 2$

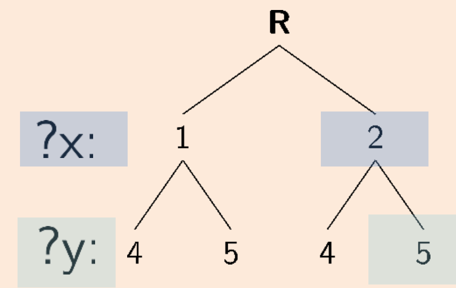
$\text{Valid}_{2,?y} = \{4, 5\}$ $y = 5$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

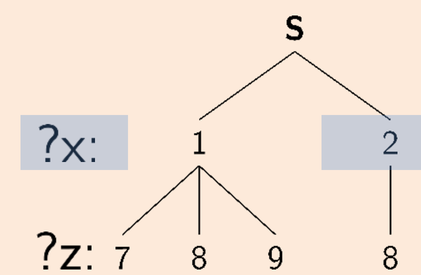
R

?x	?y
1	4
1	5
2	4
2	5



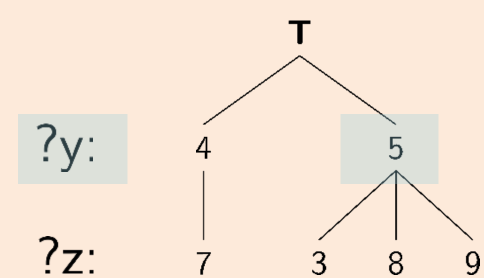
S

?x	?z
1	7
1	8
1	9
2	8



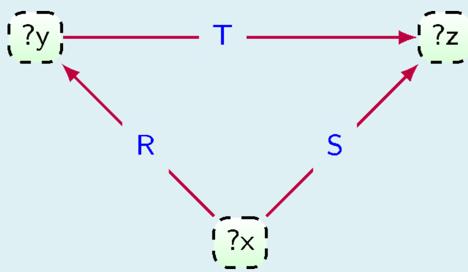
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

$\text{Valid}_{?x} = \{1,2\} \quad x = 2$

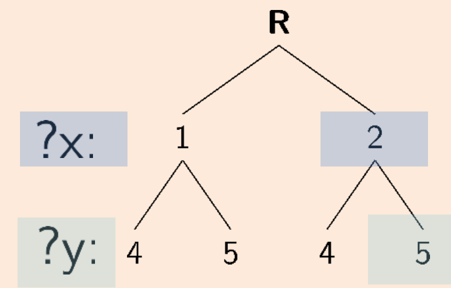
$\text{Valid}_{2,?y} = \{4,5\} \quad y = 5$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

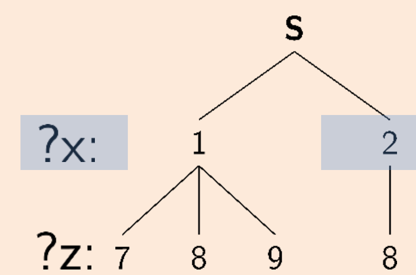
R

?x	?y
1	4
1	5
2	4
2	5



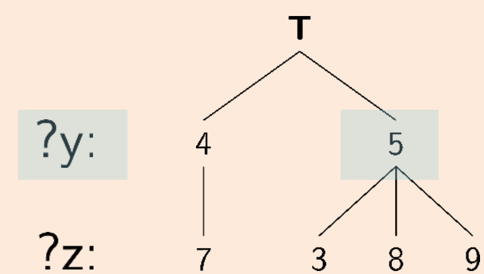
S

?x	?z
1	7
1	8
1	9
2	8



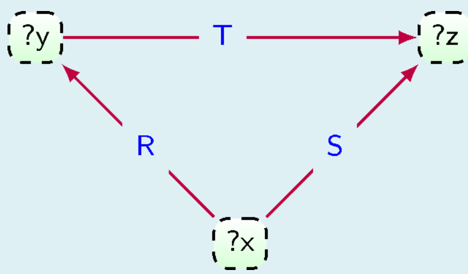
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
    $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
    for each $y \in \text{Valid}_{x,?y}$ **do**
        $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
        for each $z \in \text{Valid}_{x,y?z}$ **do**
            $\text{Sol} \leftarrow (x,y,z)$

$\text{Valid}_{?x} = \{1,2\} \quad x = 2$

$\text{Valid}_{2,?y} = \{4,5\} \quad y = 5$

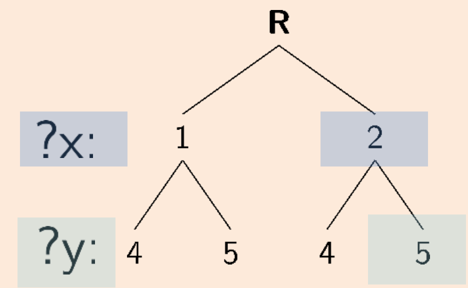
$\text{Valid}_{2,5,?z} = \{8\}$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9

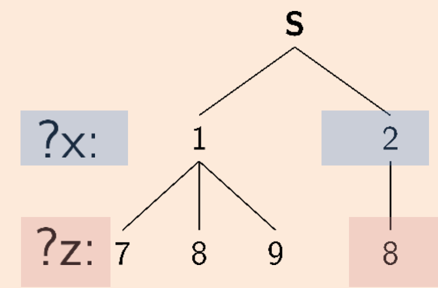
R

?x	?y
1	4
1	5
2	4
2	5



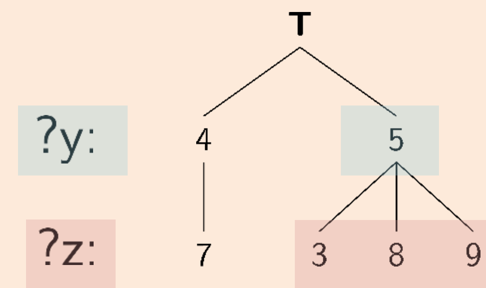
S

?x	?z
1	7
1	8
1	9
2	8



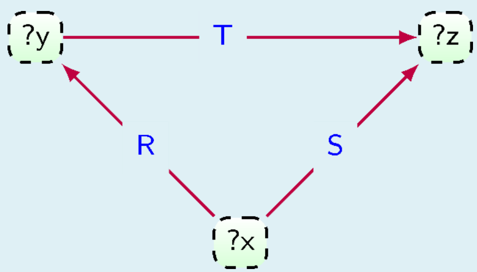
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?\mathrm{x},?\mathrm{y}) \bowtie \mathbf{S}(?\mathrm{x},?\mathrm{z}) \bowtie \mathbf{T}(?\mathrm{y},?\mathrm{z})$



GAO ?x, ?y, ?z:

$\mathrm{Valid}_{?\mathrm{x}} \leftarrow \pi_{?\mathrm{x}}(\mathbf{R}) \cap \pi_{?\mathrm{x}}(\mathbf{S})$
for each $x \in \mathrm{Valid}_{?\mathrm{x}}$ **do**
  $\mathrm{Valid}_{x,?\mathrm{y}} \leftarrow \pi_{?\mathrm{y}}(\mathbf{R}[x,?\mathrm{y}]) \cap \pi_{?\mathrm{y}}(\mathbf{T})$
  for each $y \in \mathrm{Valid}_{x,?\mathrm{y}}$ **do**
    $\mathrm{Valid}_{x,y,?\mathrm{z}} \leftarrow \pi_{?\mathrm{z}}(\mathbf{S}[x,?\mathrm{z}]) \cap \pi_{?\mathrm{z}}(\mathbf{T}[y,?\mathrm{z}])$
    for each $z \in \mathrm{Valid}_{x,y?\mathrm{z}}$ **do**
      $\mathrm{Sol} \leftarrow (x,y,z)$

$\mathrm{Valid}_{?\mathrm{x}} = \{1,2\} \quad x = 2$

$\mathrm{Valid}_{2,?\mathrm{y}} = \{4,5\} \quad y = 5$

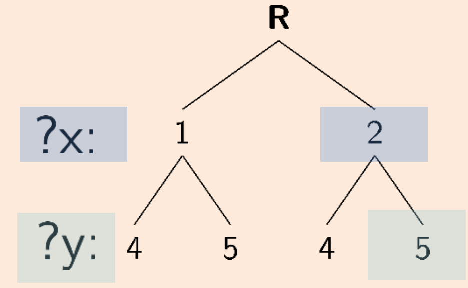
$\mathrm{Valid}_{2,5,?\mathrm{z}} = \{8\}$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9
2	5	8

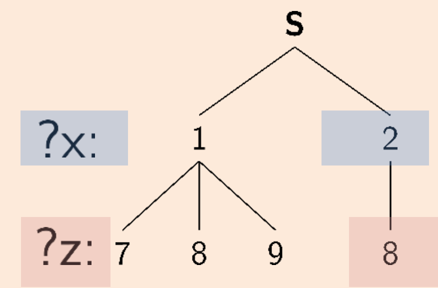
R

?x	?y
1	4
1	5
2	4
2	5



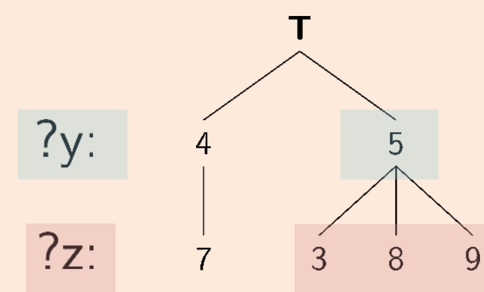
S

?x	?z
1	7
1	8
1	9
2	8



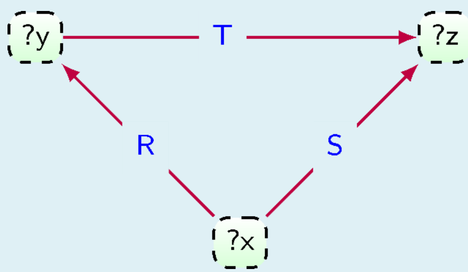
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{\text{?x}} \leftarrow \pi_{\text{?x}}(\mathbf{R}) \cap \pi_{\text{?x}}(\mathbf{S})$

for each $x \in \text{Valid}_{\text{?x}}$ **do**

  $\text{Valid}_{x,\text{?y}} \leftarrow \pi_{\text{?y}}(\mathbf{R}[x,\text{?y}]) \cap \pi_{\text{?y}}(\mathbf{T})$

  for each $y \in \text{Valid}_{x,\text{?y}}$ **do**

    $\text{Valid}_{x,y,\text{?z}} \leftarrow \pi_{\text{?z}}(\mathbf{S}[x,\text{?z}]) \cap \pi_{\text{?z}}(\mathbf{T}[y,\text{?z}])$

    for each $z \in \text{Valid}_{x,y\text{?z}}$ **do**

      $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{\text{?x}} = \{1, 2\}$ $x = 2$

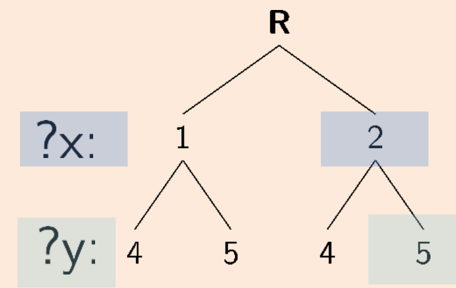
$\text{Valid}_{2,\text{?y}} = \{4, 5\}$ $y = 5$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9
2	5	8

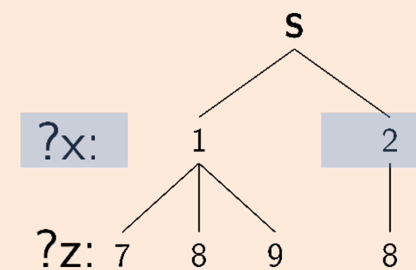
R

?x	?y
1	4
1	5
2	4
2	5



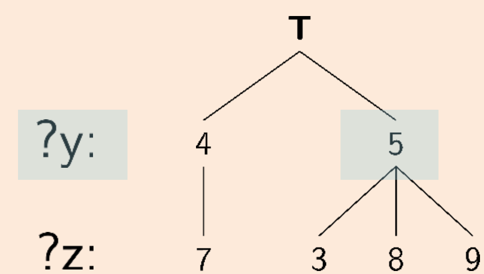
S

?x	?z
1	7
1	8
1	9
2	8



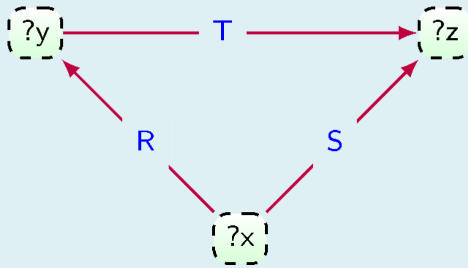
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{1, 2\} \quad x = 2$

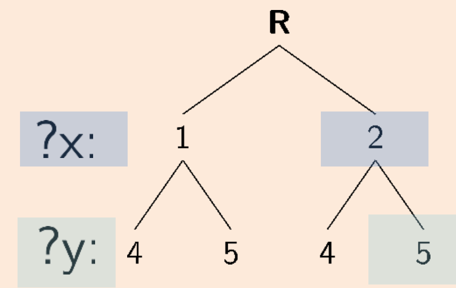
$\text{Valid}_{2,?y} = \{4, 5\} \quad y = 5$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9
2	5	8

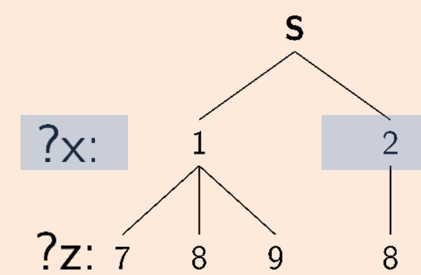
R

?x	?y
1	4
1	5
2	4
2	5



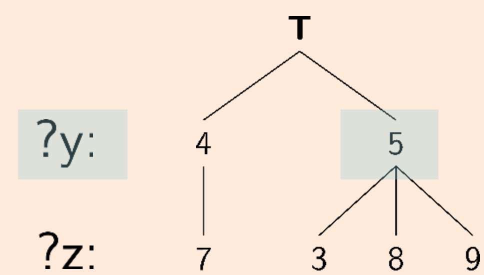
S

?x	?z
1	7
1	8
1	9
2	8



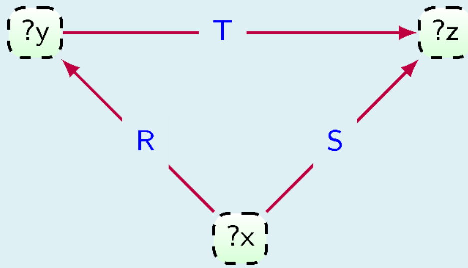
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

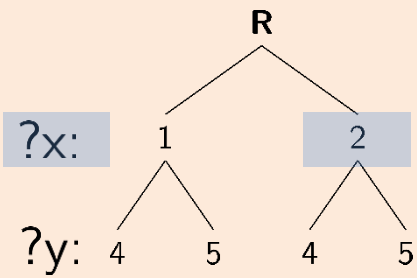
$\text{Valid}_{?x} = \{1, 2\} \quad x = 2$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9
2	5	8

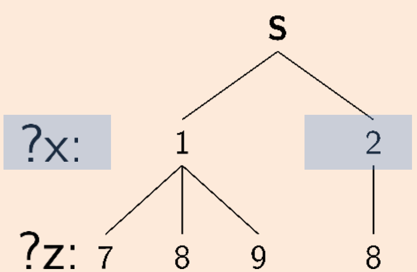
R

?x	?y
1	4
1	5
2	4
2	5



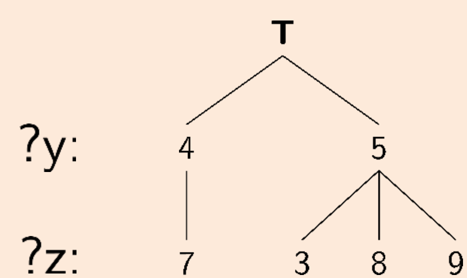
S

?x	?z
1	7
1	8
1	9
2	8



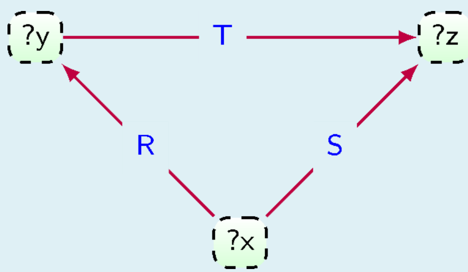
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

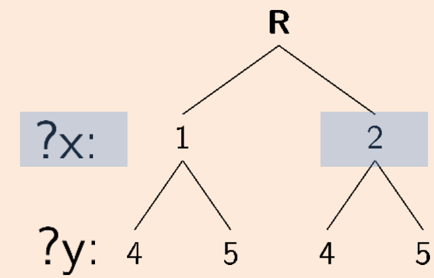
$\text{Valid}_{?x} = \{1,2\} \quad x = 2$

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9
2	5	8

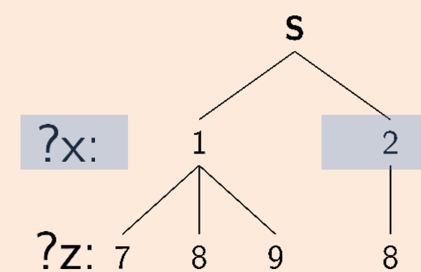
R

?x	?y
1	4
1	5
2	4
2	5



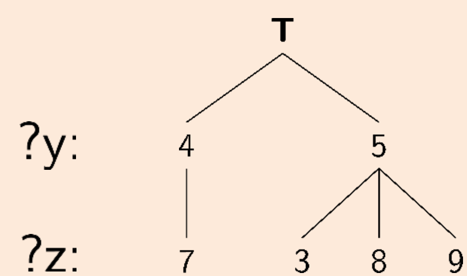
S

?x	?z
1	7
1	8
1	9
2	8



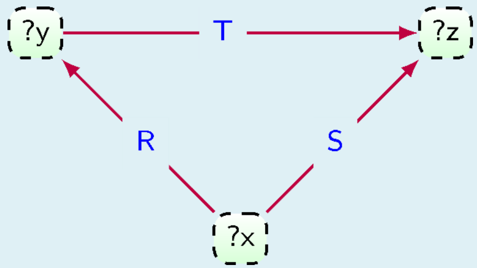
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

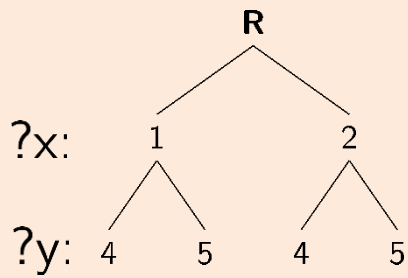
Done!

Sol

?x	?y	?z
1	4	7
1	5	8
1	5	9
2	5	8

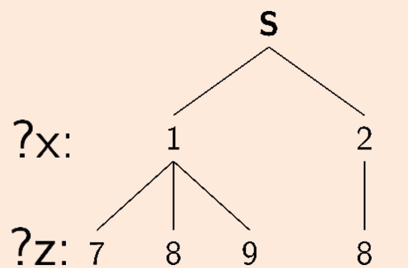
R

?x	?y
1	4
1	5
2	4
2	5



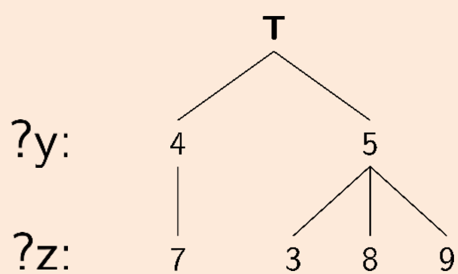
S

?x	?z
1	7
1	8
1	9
2	8



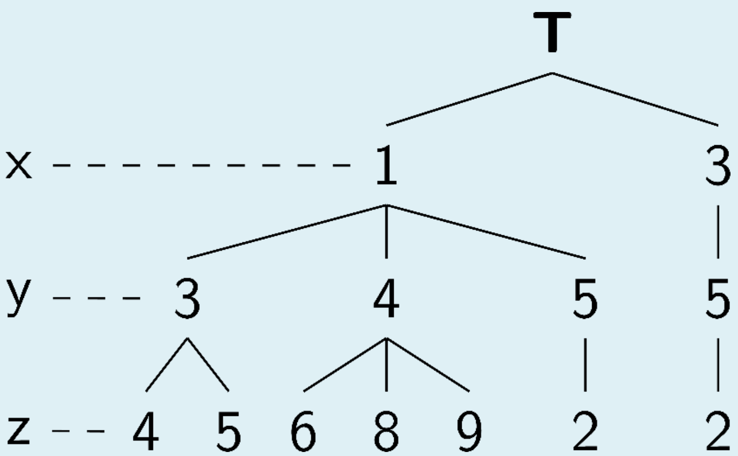
T

?y	?z
4	7
5	3
5	8
5	9



Relations are usually Tries

	T	
x	y	z
1	3	4
1	3	5
1	4	6
1	4	8
1	4	9
1	5	2
3	5	2



Most common way to store a relation?

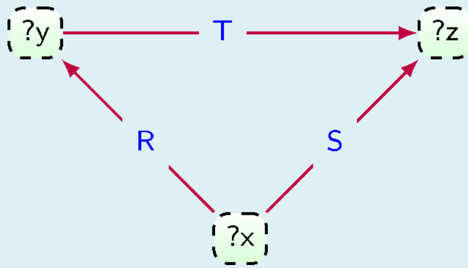
B+ tree

So we can do Leapfrog on relations

(Is it really this easy?)

Leapfrog in a triangle

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$

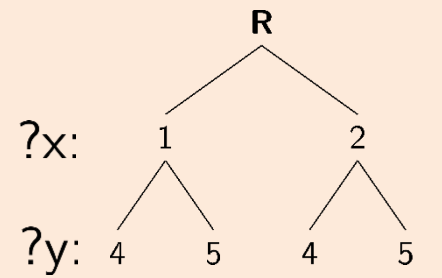


GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \text{Valid}_{?x}$ **do**
  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \text{Valid}_{x,?y}$ **do**
    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

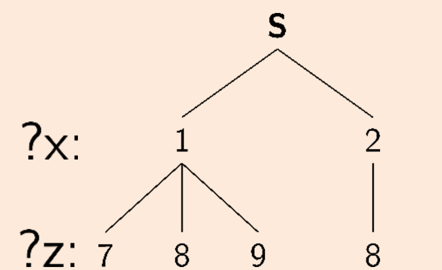
R

?x	?y
1	4
1	5
2	4
2	5



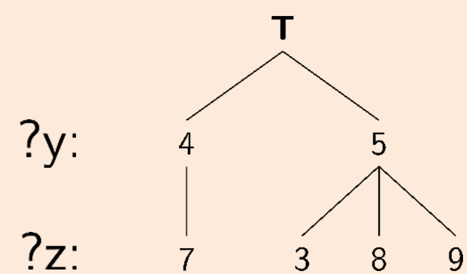
S

?x	?z
1	7
1	8
1	9
2	8



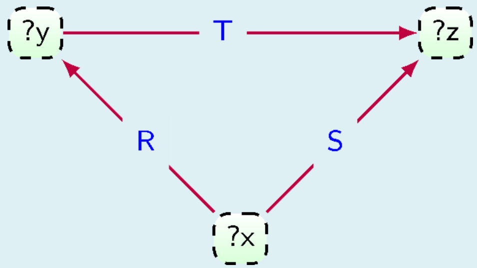
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?y,?x) \bowtie \mathbf{S}(?z,?x) \bowtie \mathbf{T}(?z,?y)$

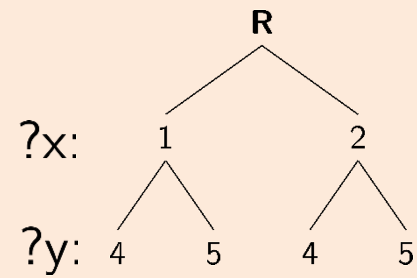


GAO ?z, ?y, ?x:

$\text{Valid}_{?z} \leftarrow \pi_{?z}(\mathbf{S}) \cap \pi_{?z}(\mathbf{T})$
for each $z \in \text{Valid}_{?z}$ **do**
$\quad \text{Valid}_{z,?y} \leftarrow \pi_{?y}(\mathbf{T}[z, ?y]) \cap \pi_{?y}(\mathbf{R})$
$\quad$ **for each** $y \in \text{Valid}_{z,?y}$ **do**
$\quad\quad \text{Valid}_{z,y,?x} \leftarrow \pi_{?x}(\mathbf{S}[z, ?x]) \cap \pi_{?x}(\mathbf{R}[y, ?x])$
$\quad\quad$ **for each** $x \in \text{Valid}_{z,y?x}$ **do**
$\quad\quad\quad \text{Sol} \leftarrow (x, y, z)$

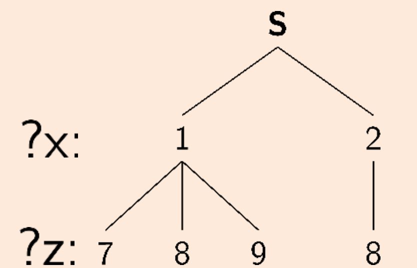
R

?x	?y
1	4
1	5
2	4
2	5



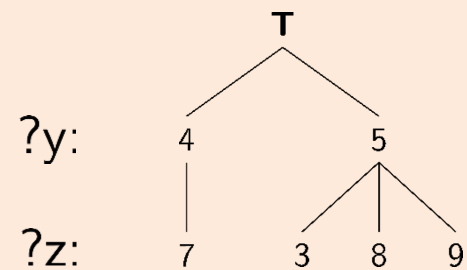
S

?x	?z
1	7
1	8
1	9
2	8



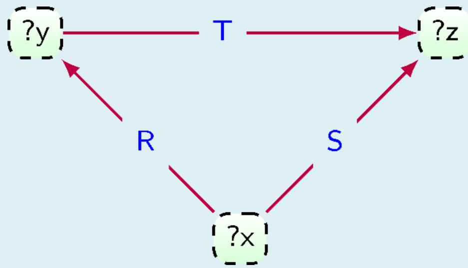
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?y,?x) \bowtie \mathbf{S}(?z,?x) \bowtie \mathbf{T}(?z,?y)$

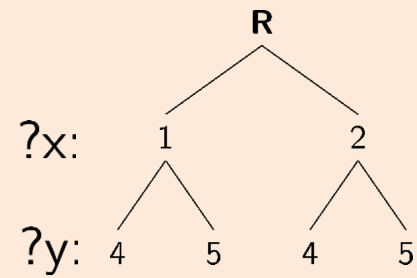


GAO ?z, ?y, ?x:

$\text{Valid}_{?z} \leftarrow \pi_{?z}(\mathbf{S}) \cap \pi_{?z}(\mathbf{T})$
for each $z \in \text{Valid}_{?z}$ **do**
  $\text{Valid}_{z,?y} \leftarrow \pi_{?y}(\mathbf{T}[z,?y]) \cap \pi_{?y}(\mathbf{R})$
  for each $y \in \text{Valid}_{z,?y}$ **do**
    $\text{Valid}_{z,y,?x} \leftarrow \pi_{?x}(\mathbf{S}[z,?x]) \cap \pi_{?x}(\mathbf{R}[y,?x])$
    for each $x \in \text{Valid}_{z,y?x}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

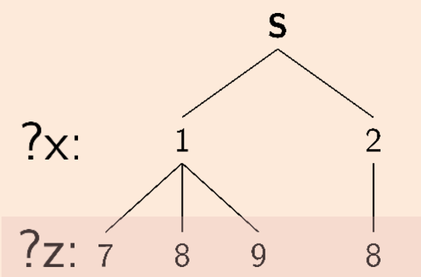
R

?x	?y
1	4
1	5
2	4
2	5



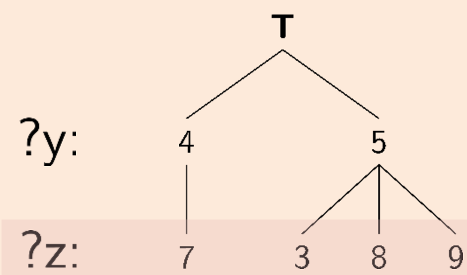
S

?x	?z
1	7
1	8
1	9
2	8



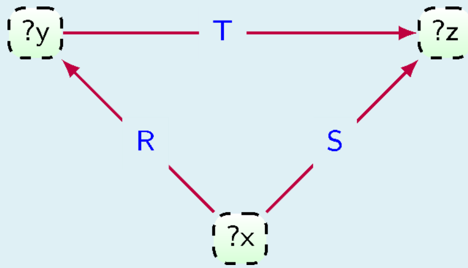
T

?y	?z
4	7
5	3
5	8
5	9



Leapfrog in a triangle

$\mathbf{R}(?y,?x) \bowtie \mathbf{S}(?z,?x) \bowtie \mathbf{T}(?z,?y)$

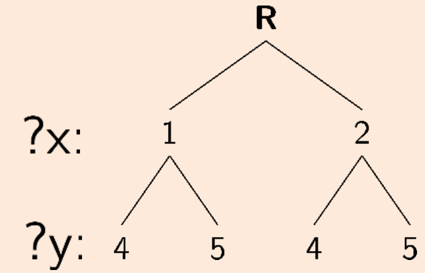


GAO ?z, ?y, ?x:

$$
\begin{array}{l}
\mathsf{Valid}_{?z} \leftarrow \pi_{?z}(\mathbf{S}) \cap \pi_{?z}(\mathbf{T}) \\
\textbf{for each } z \in \mathsf{Valid}_{?z} \textbf{ do} \\
\quad \mathsf{Valid}_{z,?y} \leftarrow \pi_{?y}(\mathbf{T}[z,?y]) \cap \pi_{?y}(\mathbf{R}) \\
\quad \textbf{for each } y \in \mathsf{Valid}_{z,?y} \textbf{ do} \\
\qquad \mathsf{Valid}_{z,y,?x} \leftarrow \pi_{?x}(\mathbf{S}[z,?x]) \cap \pi_{?x}(\mathbf{R}[y,?x]) \\
\qquad \textbf{for each } x \in \mathsf{Valid}_{z,y?x} \textbf{ do} \\
\qquad\quad \mathsf{Sol} \leftarrow (x,y,z)
\end{array}
$$

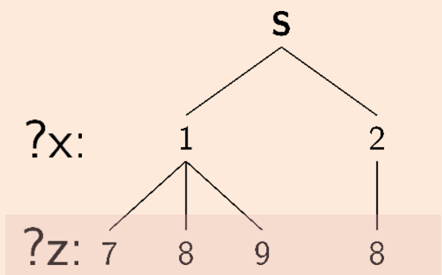
R

?x	?y
1	4
1	5
2	4
2	5

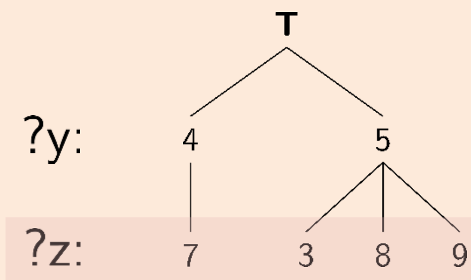


S

?x	?z
1	7
1	8
1	9
2	8



?y	?z
4	7
5	3
5	8
5	9

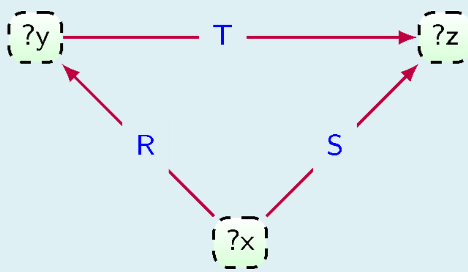


Cannot do efficient intersection!

(We need a Trie starting with ?z)

Leapfrog in a triangle

R(?y,?x) ⋈ **S**(?z,?x) ⋈ **T**(?z,?y)



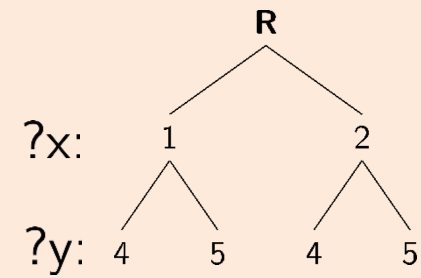
GAO ?z, ?y, ?x:

$\text{Valid}_{?z} \leftarrow \pi_{?z}(\mathbf{S}) \cap \pi_{?z}(\mathbf{T})$
for each $z \in \text{Valid}_{?z}$ **do**
 $\text{Valid}_{z,?y} \leftarrow \pi_{?y}(\mathbf{T}[z, ?y]) \cap \pi_{?y}(\mathbf{R})$
 for each $y \in \text{Valid}_{z,?y}$ **do**
  $\text{Valid}_{z,y,?x} \leftarrow \pi_{?x}(\mathbf{S}[z, ?x]) \cap \pi_{?x}(\mathbf{R}[y, ?x])$
  for each $x \in \text{Valid}_{z,y?x}$ **do**
   $\text{Sol} \leftarrow (x, y, z)$

- To support any GAO:
  - We need all the permutations of the attributes
  - Table with n attributes = n! permutations

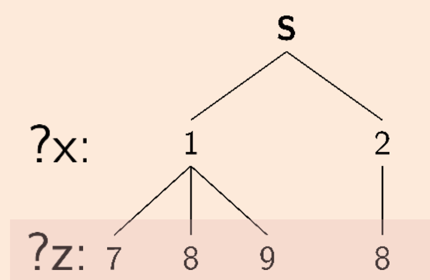
R

?x	?y
1	4
1	5
2	4
2	5



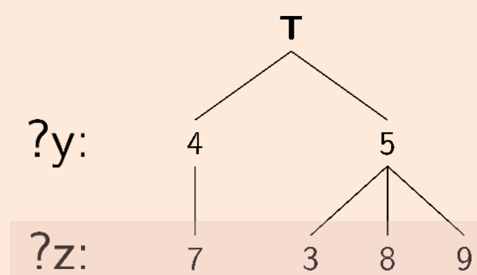
S

?x	?z
1	7
1	8
1	9
2	8



T

?y	?z
4	7
5	3
5	8
5	9



How many permutations?

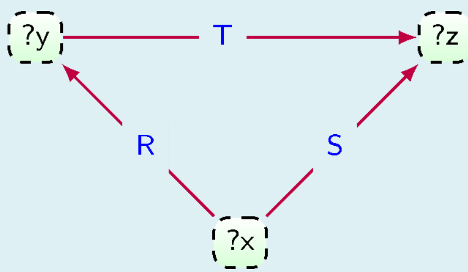
- This can get expensive
  - Need many permutations
  - Many many many permutations
  - Basically all column orderings of your tables
  - 3! = 6 for RDF
  - 4! + 2! + 3! = too many for PGs

RDF **Triples**(subject, predicate, object)

PGs
Connections(src, label, tgt, <u>eId</u>)
Labels(objectId, label)
Properties(<u>objectId</u>, <u>key</u>, value)

Leapfrog is "sensitive"

$\mathbf{R}(?\mathrm{x},?\mathrm{y}) \bowtie \mathbf{S}(?\mathrm{x},?\mathrm{z}) \bowtie \mathbf{T}(?\mathrm{y},?\mathrm{z})$



GAO ?x, ?y, ?z:

$\mathrm{Valid}_{?\mathrm{x}} \leftarrow \pi_{?\mathrm{x}}(\mathbf{R}) \cap \pi_{?\mathrm{x}}(\mathbf{S})$
for each $x \in \mathrm{Valid}_{?\mathrm{x}}$ **do**
  $\mathrm{Valid}_{x,?\mathrm{y}} \leftarrow \pi_{?\mathrm{y}}(\mathbf{R}[x,?\mathrm{y}]) \cap \pi_{?\mathrm{y}}(\mathbf{T})$
  for each $y \in \mathrm{Valid}_{x,?\mathrm{y}}$ **do**
    $\mathrm{Valid}_{x,y,?\mathrm{z}} \leftarrow \pi_{?\mathrm{z}}(\mathbf{S}[x,?\mathrm{z}]) \cap \pi_{?\mathrm{z}}(\mathbf{T}[y,?\mathrm{z}])$
    for each $z \in \mathrm{Valid}_{x,y?\mathrm{z}}$ **do**
      $\mathrm{Sol} \leftarrow (x,y,z)$

R

?x	?y
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_n	y_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

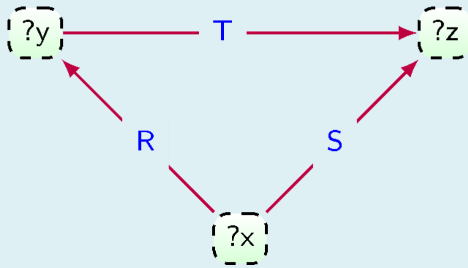
T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1,y_1,z_1)\}$$

Leapfrog is "sensitive"

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\mathsf{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in \mathsf{Valid}_{?x}$ **do**
  $\mathsf{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in \mathsf{Valid}_{x,?y}$ **do**
    $\mathsf{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \mathsf{Valid}_{x,y?z}$ **do**
      $\mathsf{Sol} \leftarrow (x, y, z)$

$$\mathsf{Valid}_{?x} = \{x_1, \ldots, x_n\}$$

R

?x	?y
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_n	y_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

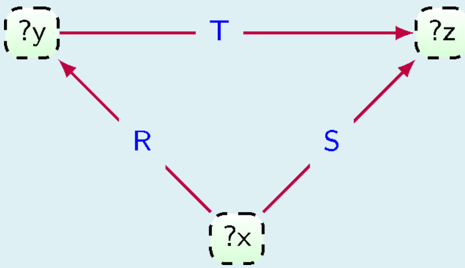
T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$$

Leapfrog is "sensitive"

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?x, ?y, ?z:

$\text{Valid}_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$

for each $x \in \text{Valid}_{?x}$ **do**

  $\text{Valid}_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$

  for each $y \in \text{Valid}_{x,?y}$ **do**

    $\text{Valid}_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$

    for each $z \in \text{Valid}_{x,y?z}$ **do**

      $\text{Sol} \leftarrow (x, y, z)$

$\text{Valid}_{?x} = \{x_1, \ldots, x_n\}$

… do something for each x_i

R

?x	?y
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_n	y_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

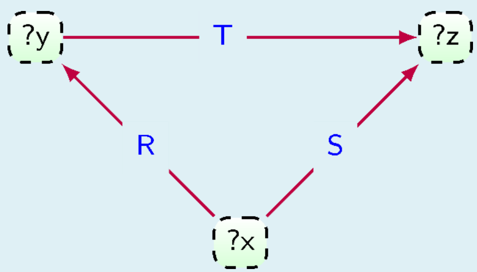
T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$

Leapfrog is "sensitive"

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?y, ?x, ?z:

$\text{Valid}_{?\text{x}} \leftarrow \pi_{?\text{x}}(\mathbf{R}) \cap \pi_{?\text{x}}(\mathbf{S})$
for each $x \in \text{Valid}_{?\text{x}}$ **do**
    $\text{Valid}_{x,?\text{y}} \leftarrow \pi_{?\text{y}}(\mathbf{R}[x, ?\text{y}]) \cap \pi_{?\text{y}}(\mathbf{T})$
    for each $y \in \text{Valid}_{x,?\text{y}}$ **do**
        $\text{Valid}_{x,y,?\text{z}} \leftarrow \pi_{?\text{z}}(\mathbf{S}[x, ?\text{z}]) \cap \pi_{?\text{z}}(\mathbf{T}[y, ?\text{z}])$
        for each $z \in \text{Valid}_{x,y?\text{z}}$ **do**
            $\text{Sol} \leftarrow (x, y, z)$

R

?x	?y
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_n	y_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

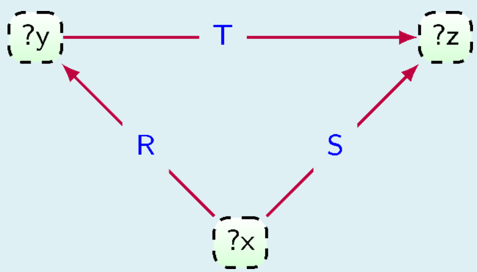
T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$$

Leapfrog is "sensitive"

$\mathbf{R}(?x,?y) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?y, ?x, ?z:

$\mathrm{Valid}_{?y} \leftarrow \pi_{?y}(\mathbf{R}) \cap \pi_{?y}(\mathbf{T})$
for each $y \in \mathrm{Valid}_{?y}$ **do**
  $\mathrm{Valid}_{y,?x} \leftarrow \pi_{?x}(\mathbf{R}[y,?x]) \cap \pi_{?x}(\mathbf{S})$
  for each $x \in \mathrm{Valid}_{y,?x}$ **do**
    $\mathrm{Valid}_{y,x,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \mathrm{Valid}_{x,y?z}$ **do**
      $\mathrm{Sol} \leftarrow (x,y,z)$

R

?x	?y
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_n	y_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

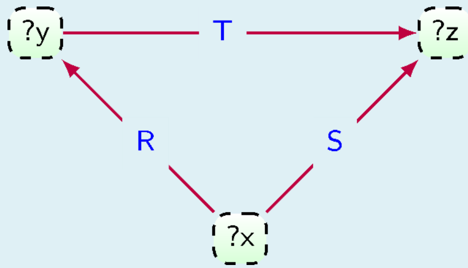
T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$$

Leapfrog is "sensitive"

$\mathbf{R}(?y,?x) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$



GAO ?y, ?x, ?z:

$\text{Valid}_{?y} \leftarrow \pi_{?y}(\mathbf{R}) \cap \pi_{?y}(\mathbf{T})$
for each $y \in \text{Valid}_{?y}$ **do**
  $\text{Valid}_{y,?x} \leftarrow \pi_{?x}(\mathbf{R}[y, ?x]) \cap \pi_{?x}(\mathbf{S})$
  for each $x \in \text{Valid}_{y,?x}$ **do**
    $\text{Valid}_{y,x,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

R

?y	?x
y_1	x_1
y_2	x_2
$\vdots$	$\vdots$
y_n	x_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

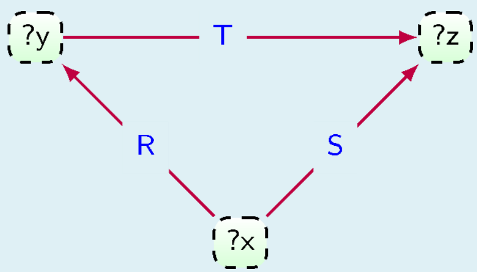
T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$$

Leapfrog is "sensitive"

$\mathbf{R}(?y,?x) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?y, ?x, ?z:

$\text{Valid}_{?y} \leftarrow \pi_{?y}(\mathbf{R}) \cap \pi_{?y}(\mathbf{T})$
for each $y \in \text{Valid}_{?y}$ **do**
  $\text{Valid}_{y,?x} \leftarrow \pi_{?x}(\mathbf{R}[y, ?x]) \cap \pi_{?x}(\mathbf{S})$
  for each $x \in \text{Valid}_{y,?x}$ **do**
    $\text{Valid}_{y,x,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

R

?y	?x
y_1	x_1
y_2	x_2
⋮	⋮
y_n	x_n

S

?x	?z
x_1	z_1
x_2	z_2
⋮	⋮
x_n	z_n

T

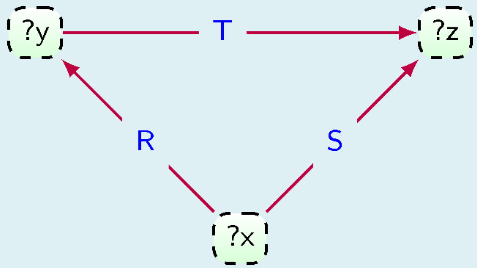
?y	?z
y_1	z_1
y_1	z_2
⋮	⋮
y_1	z_n

$\text{Valid}_{?y} = \{y_1\}$

$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$

Leapfrog is "sensitive"

$\mathbf{R}(?y,?x) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y,\ ?z)$



GAO ?y, ?x, ?z:

$\text{Valid}_{?y} \leftarrow \pi_{?y}(\mathbf{R}) \cap \pi_{?y}(\mathbf{T})$
for each $y \in \text{Valid}_{?y}$ **do**
  $\text{Valid}_{y,?x} \leftarrow \pi_{?x}(\mathbf{R}[y, ?x]) \cap \pi_{?x}(\mathbf{S})$
  for each $x \in \text{Valid}_{y,?x}$ **do**
    $\text{Valid}_{y,x,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

R

?y	?x
y_1	x_1
y_2	x_2
$\vdots$	$\vdots$
y_n	x_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

T

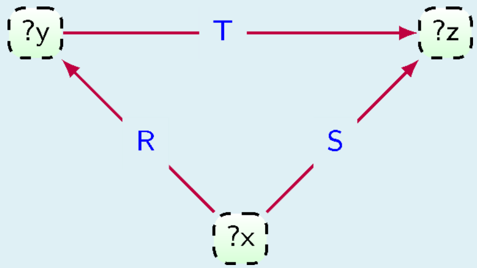
?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$\text{Valid}_{?y} = \{y_1\}$

$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$

Leapfrog is "sensitive"

$\mathbf{R}(\text{?y,?x}) \bowtie \mathbf{S}(\text{?x,?z}) \bowtie \mathbf{T}(\text{?y, ?z})$



GAO ?y, ?x, ?z:

$\text{Valid}_{?\text{y}} \leftarrow \pi_{?\text{y}}(\mathbf{R}) \cap \pi_{?\text{y}}(\mathbf{T})$
for each $y \in \text{Valid}_{?\text{y}}$ **do**
  $\text{Valid}_{y,?\text{x}} \leftarrow \pi_{?\text{x}}(\mathbf{R}[y, ?\text{x}]) \cap \pi_{?\text{x}}(\mathbf{S})$
  for each $x \in \text{Valid}_{y,?\text{x}}$ **do**
    $\text{Valid}_{y,x,?\text{z}} \leftarrow \pi_{?\text{z}}(\mathbf{S}[x, ?\text{z}]) \cap \pi_{?\text{z}}(\mathbf{T}[y, ?\text{z}])$
    for each $z \in \text{Valid}_{x,y?\text{z}}$ **do**
      $\text{Sol} \leftarrow (x, y, z)$

$$\text{Valid}_{?\text{y}} = \{y_1\}$$

$$\text{Valid}_{y_1,?\text{x}} = \{x_1\}$$

R

?y	?x
y_1	x_1
y_2	x_2
$\vdots$	$\vdots$
y_n	x_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

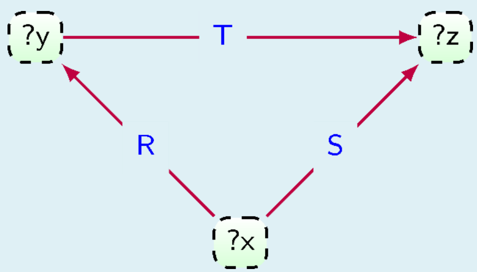
T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$$

Leapfrog is "sensitive"

R(?y,?x) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?y, ?x, ?z:

$\text{Valid}_{?\text{y}} \leftarrow \pi_{?\text{y}}(\mathbf{R}) \cap \pi_{?\text{y}}(\mathbf{T})$

for each $y \in \text{Valid}_{?\text{y}}$ **do**

$\quad \text{Valid}_{y,?\text{x}} \leftarrow \pi_{?\text{x}}(\mathbf{R}[y, ?\text{x}]) \cap \pi_{?\text{x}}(\mathbf{S})$

$\quad$ **for each** $x \in \text{Valid}_{y,?\text{x}}$ **do**

$\quad\quad \text{Valid}_{y,x,?\text{z}} \leftarrow \pi_{?\text{z}}(\mathbf{S}[x, ?\text{z}]) \cap \pi_{?\text{z}}(\mathbf{T}[y, ?\text{z}])$

$\quad\quad$ **for each** $z \in \text{Valid}_{x,y?\text{z}}$ **do**

$\quad\quad\quad \text{Sol} \leftarrow (x, y, z)$

$$\text{Valid}_{?\text{y}} = \{y_1\}$$

$$\text{Valid}_{y_1,?\text{x}} = \{x_1\}$$

R

?y	?x
y_1	x_1
y_2	x_2
$\vdots$	$\vdots$
y_n	x_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

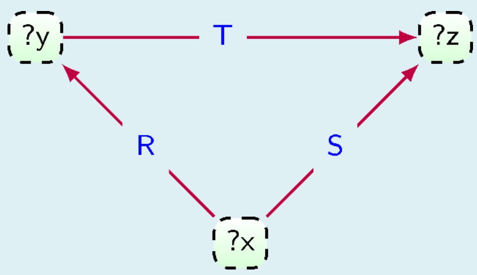
T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$$

Leapfrog is "sensitive"

$\mathbf{R}(?y,?x) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?y, ?x, ?z:

$\text{Valid}_{?y} \leftarrow \pi_{?y}(\mathbf{R}) \cap \pi_{?y}(\mathbf{T})$
for each $y \in \text{Valid}_{?y}$ **do**
$\quad \text{Valid}_{y,?x} \leftarrow \pi_{?x}(\mathbf{R}[y, ?x]) \cap \pi_{?x}(\mathbf{S})$
$\quad$ **for each** $x \in \text{Valid}_{y,?x}$ **do**
$\quad\quad \text{Valid}_{y,x,?z} \leftarrow \pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
$\quad\quad$ **for each** $z \in \text{Valid}_{x,y?z}$ **do**
$\quad\quad\quad \text{Sol} \leftarrow (x, y, z)$

R

?y	?x
y_1	x_1
y_2	x_2
$\vdots$	$\vdots$
y_n	x_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\text{Valid}_{?y} = \{y_1\}$$

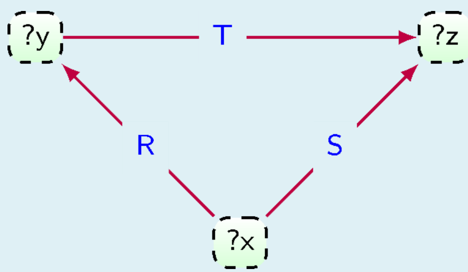
$$\text{Valid}_{y_1,?x} = \{x_1\}$$

$$\text{Valid}_{y_1,x_1,?z} = \{z_1\}$$

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$$

Leapfrog is "sensitive"

$\mathbf{R}(?y,?x) \bowtie \mathbf{S}(?x,?z) \bowtie \mathbf{T}(?y, ?z)$



GAO ?y, ?x, ?z:

$\text{Valid}_{?y} \leftarrow \pi_{?y}(\mathbf{R}) \cap \pi_{?y}(\mathbf{T})$
for each $y \in \text{Valid}_{?y}$ **do**
  $\text{Valid}_{y,?x} \leftarrow \pi_{?x}(\mathbf{R}[y,?x]) \cap \pi_{?x}(\mathbf{S})$
  for each $x \in \text{Valid}_{y,?x}$ **do**
    $\text{Valid}_{y,x,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in \text{Valid}_{x,y?z}$ **do**
      $\text{Sol} \leftarrow (x,y,z)$

R

?y	?x
y_1	x_1
y_2	x_2
$\vdots$	$\vdots$
y_n	x_n

S

?x	?z
x_1	z_1
x_2	z_2
$\vdots$	$\vdots$
x_n	z_n

T

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

$$\text{Valid}_{?y} = \{y_1\}$$

$$\text{Valid}_{y_1,?x} = \{x_1\}$$

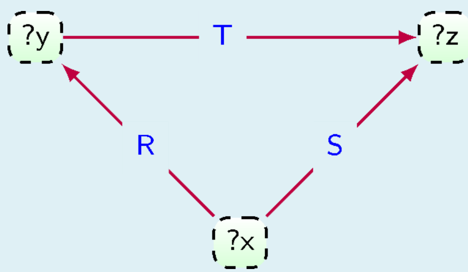
$$\text{Valid}_{y_1,x_1,?z} = \{z_1\}$$

Optimal!

$$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$$

Leapfrog is "sensitive"

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)



GAO ?x, ?y, ?z:

Valid$_{?x}$ ← $\pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in$ Valid$_{?x}$ **do**
  Valid$_{x,?y}$ ← $\pi_{?y}(\mathbf{R}[x, ?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in$ Valid$_{x,?y}$ **do**
    Valid$_{x,y,?z}$ ← $\pi_{?z}(\mathbf{S}[x, ?z]) \cap \pi_{?z}(\mathbf{T}[y, ?z])$
    for each $z \in$ Valid$_{x,y?z}$ **do**
      Sol ← (x, y, z)

Valid$_{?x}$ $= \{x_1, \ldots, x_n\}$

… do something for each x_i

R

?x	?y
x_1	

S

?x	?z

Hmm... are you not supposed to be optimal?

?y	?z
y_1	z_1
y_1	z_2
⋮	⋮
y_1	z_n

$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$

Leapfrog is "sensitive"

R(?x,?y) ⋈ **S**(?x,?z) ⋈ **T**(?y, ?z)

I'm optimal in the worst case!
(and this is not the worst case)

Hmm... are you not supposed to be optimal?

R

?x	?y
x_1	

S

?x	?z

?y	?z
y_1	z_1
y_1	z_2
$\vdots$	$\vdots$
y_1	z_n

GAO ?x, ?y, ?z:

Valid$_{?x} \leftarrow \pi_{?x}(\mathbf{R}) \cap \pi_{?x}(\mathbf{S})$
for each $x \in$ Valid$_{?x}$ **do**
  Valid$_{x,?y} \leftarrow \pi_{?y}(\mathbf{R}[x,?y]) \cap \pi_{?y}(\mathbf{T})$
  for each $y \in$ Valid$_{x,?y}$ **do**
    Valid$_{x,y,?z} \leftarrow \pi_{?z}(\mathbf{S}[x,?z]) \cap \pi_{?z}(\mathbf{T}[y,?z])$
    for each $z \in$ Valid$_{x,y?z}$ **do**
      Sol $\leftarrow (x,y,z)$

Valid$_{?x} = \{x_1, \ldots, x_n\}$

... do something for each x_i

$\mathbf{R} \bowtie \mathbf{S} \bowtie \mathbf{T} = \{(x_1, y_1, z_1)\}$

Worst-case optimal joins wrapup

- Storage can be expensive
  - 1.8TB for full Wikidata (4 permutations, B+ trees)
  - Simple compression of B+trees ~ 900GB
  - Compressed representation possible ([Ring, QDags])
    - These simulate all the permutations

- Cashing reusable things migh be a bad idea
  - For Truthy this worked great
  - But in full WikiData it gets to 10GB

- Elephant in the room (no, it's not Postgres):
  - 4 permutations or more need to be updated/versioned
  - Still works decent in our setup, but is expensive

Worst-case optimal joins wrapup

- Guarantee to run in the best time in the worst case!
  - Basically never more steps then the number of query results
  - Outperform classical pairwise join plans on „worst” instances
- Benefits of LeapfrogTriejoin
  - Works with B+trees
  - Works with MVCC/SI and updates out of the box

Worst-case optimal joins – our take

- RDF:
  - SPO, POS, OSP, PSO
- PGs:
  - eId is key – stays last, so same orders as RDF
- Allows answering all queries where edge label is known!
  - These are usually the ones you would be interested in
  - Since search is not done in the void
- For missing permutations:
  - Cost-based implementations (Sellinger and Greedy)

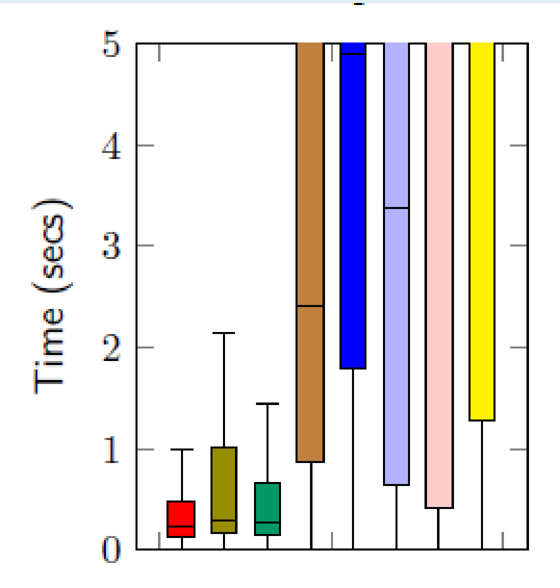
Is Leapfrog/WCO any good? (apples to apples)

- Now we can test different algorithms in the same engine
  - Important: data on disk buffered to main memory

- Wikidata-based benchmark:
  - 1.25B edges
  - 300M nodes
  - 60000 edge labels
  - Queries from the public log (so real ones)
    - Only non-bot queries
    - Eliminating duplicates (check [WDBENCH])
    - 436 complex joins
  - Start with a cold engine, data loaded as needed

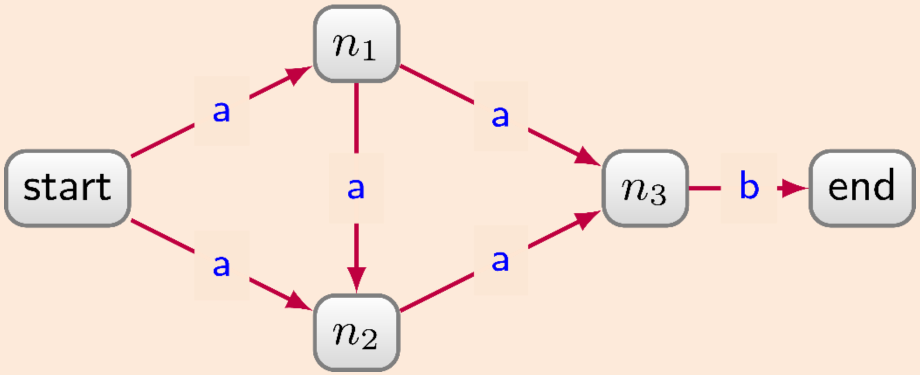
Is Leapfrog/WCO any good? (apples to apples)

Engine	Supported	Error	Timeouts	Average	Median
MillenniumDB LF	436	0	0	4.84	0.24
MillenniumDB GR	436	0	1	10.19	0.30
MillenniumDB SL	436	0	1	10.04	0.27
	436	0	3	31.79	2.42
	426	10	0	35.43	4.90
Jena LF	418	18	0	16.78	3.39
	436	0	0	7.87	5.11
	405	31	0	75.55	6.84

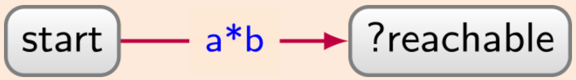


Part 3:
Evaluation of Path Queries

What does a path query return?



RPQ:



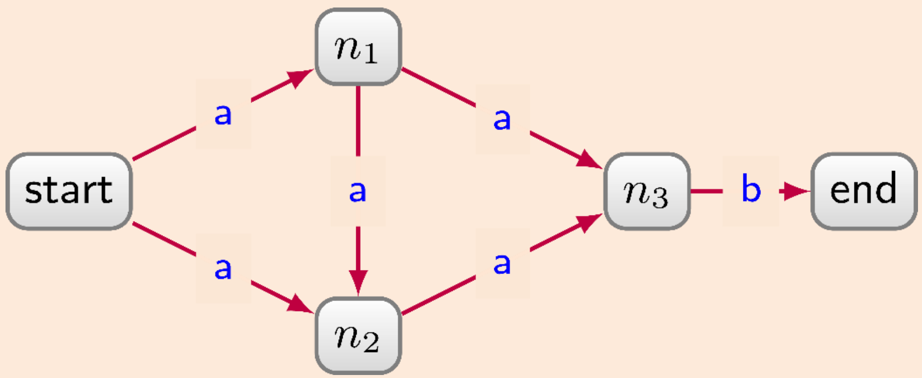
Result:

?reachable
end

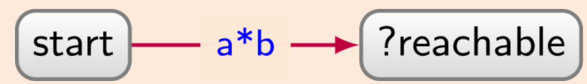
All nodes:

- Reachable from **start** in our graph
- Via a **path**
- Whose edge label matches **a*b**

What does a path query return?



RPQ:



What if I also want the path?

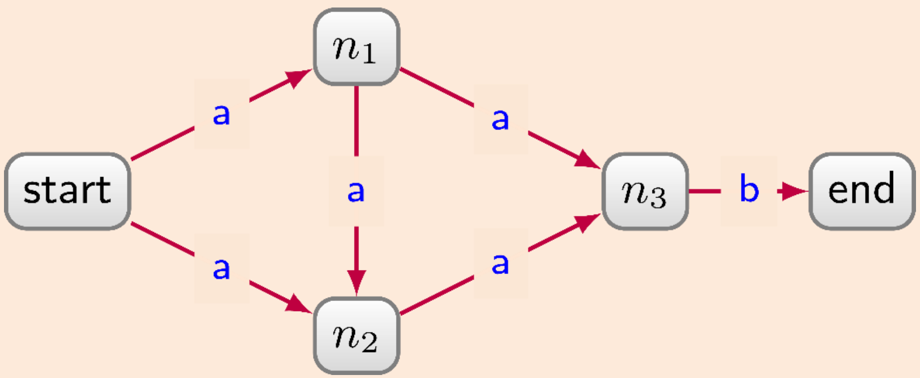
Result:

?reachable
end

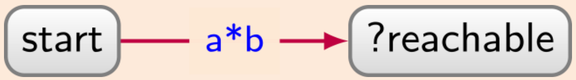
All nodes:

- Reachable from **start** in our graph
- Via a **path**
- Whose edge label matches **a*b**

What does a path query return?



RPQ:



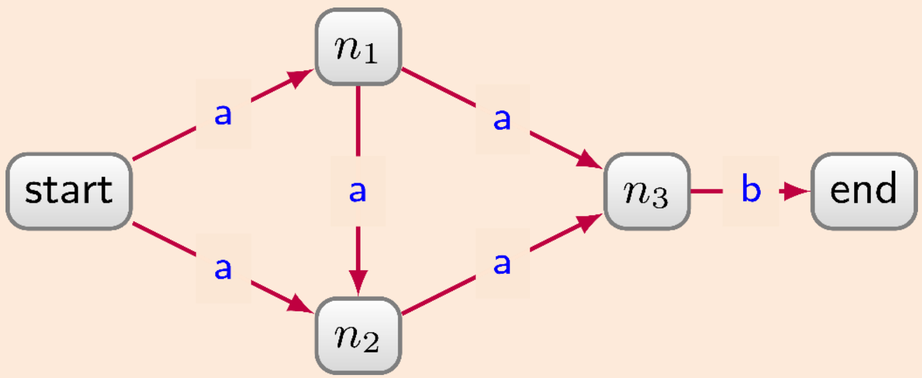
Result:

?reachable
end

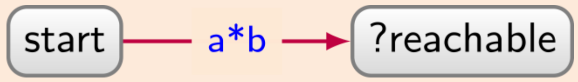
I also want the path:

- **Path #1: start→n1→n3→end**
- **Path #2: start→n1→n2→n3→end**
- **Path #3: start→n2→n3→end**

What does a path query return?



RPQ:



Which one?

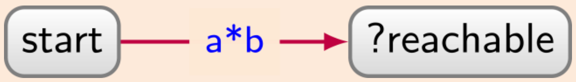
Result:

?reachable
end

I also want the path:

- **Path #1: start→n1→n3→end**
- **Path #2: start→n1→n2→n3→end**
- **Path #3: start→n2→n3→end**

What GQL proposes – you tell me



$$?p = \text{ANY WALK } \texttt{(start)=[a}^{*}\texttt{b]=>(?reachable)}$$

$$?p = \text{ANY SHORTEST WALK } \texttt{(start)=[a}^{*}\texttt{b]=>(?reachable)}$$

$$?p = \text{ALL SHORTEST WALK } \texttt{(start)=[a}^{*}\texttt{b]=>(?reachable)}$$

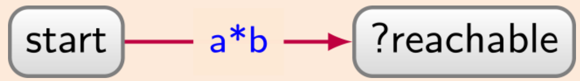
Result:

?reachable
end

I also want the path:

- **Path #1: start→n1→n3→end**
- **Path #2: start→n1→n2→n3→end**
- **Path #3: start→n2→n3→end**

What GQL proposes – you tell me



$?p =$ ANY WALK (start)=[a*b]=>(?reachable)

$?p =$ ANY SHORTE T WALK (start)=[a*b]=>(?reachable)

$?p =$ ALL SHORTE LK (start)=[a*b]=>(?reachable)

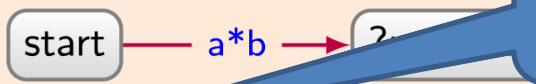
Why WALK?
Mathematicians call a path a walk

Result:

?reachable
end

- **Path #2: start→n1→n2→n3→end**
- **Path #3: start→n2→n3→end**

What GQL proposes – you tell me



For each ?reachable one path (nondeterministic)

$?p =$ ANY WALK (start)=[a*b]=>(?reachable)

$?p =$ ANY SHORTEST WALK (start)=[a*b]=>(?reachable)

$?p =$ ALL SHORTEST WALK (start)=[a*b]=>(?reachable)

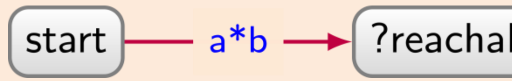
Result:

?reachable
end

I also want the path:

- **Path #1: start→n1→n3→end**
- **Path #2: start→n1→n2→n3→end**
- **Path #3: start→n2→n3→end**

What GQL proposes – you tell me



For each ?reachable one shortest path (nondeterministic)

$?p =$ ANY WALK `(start)=[a*b]=>(?reachable)`

$?p =$ ANY SHORTEST WALK `(start)=[a*b]=>(?reachable)`

$?p =$ ALL SHORTEST WALK `(start)=[a*b]=>(?reachable)`

Result:

?reachable
end

I also want the path:

- **Path #1: start→n1→n3→end**
- **Path #2: start→n1→n2→n3→end**
- **Path #3: $\boxed{\text{start→n2→n3→end}}$**

What GQL proposes – you tell me

$\text{start} \xrightarrow{a^*b} \text{?reachable}$

For each ?reachable
all shortest paths

$?p =$ ANY WALK `(start)=[a*b]=>(?reachable)`

$?p =$ ANY SHORTEST WALK `(start)=[a*b]=>(?reachable)`

$?p =$ ALL SHORTEST WALK `(start)=[a*b]=>(?reachable)`

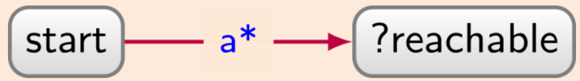
Result:

?reachable
end

I also want the path:

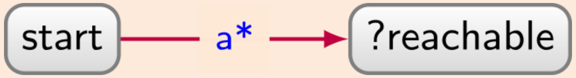
- **Path #1: start→n1→n3→end**
- **Path #2: start→n1→n2→n3→end**
- **Path #3: start→n2→n3→end**

This would be too much



$?p =$ ALL WALK `(start)=[a*]=>(?reachable)`

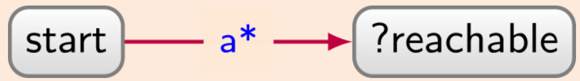
This would be too much



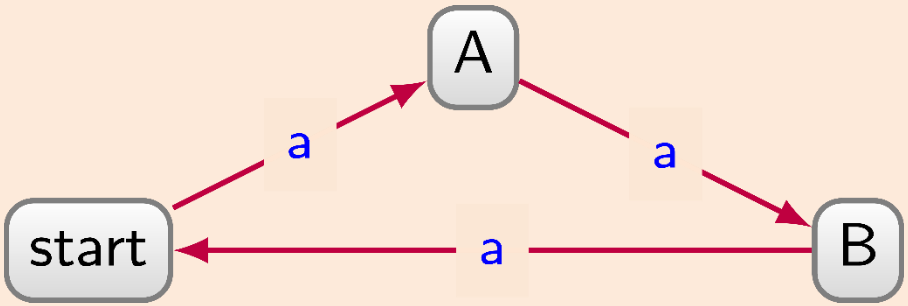
$$?p = \textsf{ALL WALK}\ \texttt{(start)=[a}^{*}\texttt{]=>(?reachable)}$$

For each ?reachable all paths

This would be too much



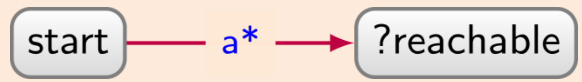
$$?p = \mathsf{ALL\ WALK}\ \texttt{(start)=[a}^{*}\texttt{]=>(?reachable)}$$



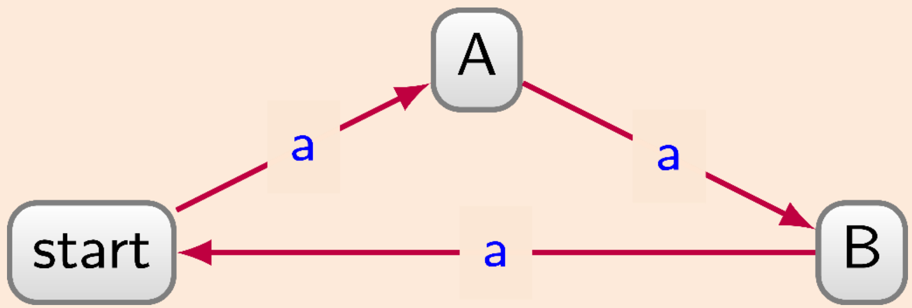
A is reachable from **start** by:

- start→A
- start→A→B→start→A
- start→A→B→start→A→B→start→A
- ...

This would be too much



$$?p = \text{ALL WALK } \texttt{(start)=[a}^{*}\texttt{]=>(?reachable)}$$

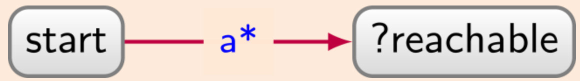


A is reachable from **start** by:

- start→A
- start→A→B→start→A
- start→A→B→start→A→B→start→A
- ...

Infinite ☹
(NOT GOOD FOR YOUR PC)

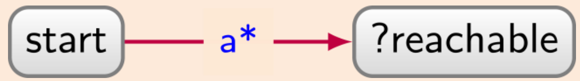
But this is OK – ALL SIMPLE



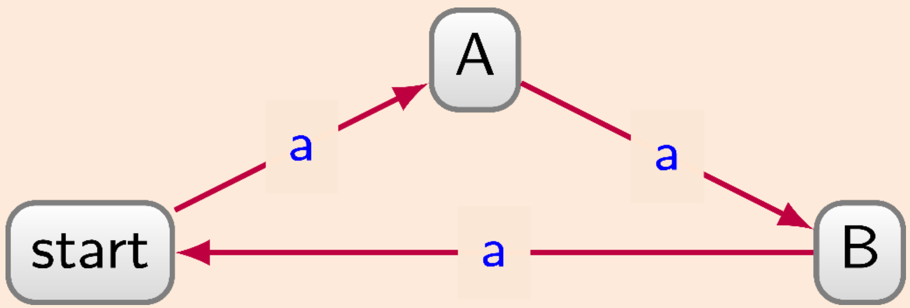
$$?p = \mathsf{SIMPLE}\ \texttt{(start)=[a}^{*}\texttt{]=>(?reachable)}$$

No node is repeated
in the path

SIMPLE Path semantics



$$?p = \mathsf{SIMPLE}\ \texttt{(start)=[a}^{*}\texttt{]=>(?reachable)}$$

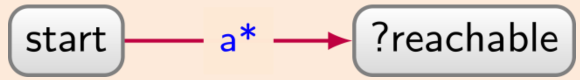


A is reachable from **start** by:

- start→A
- start→A→B→start→A

(No infinite looping)

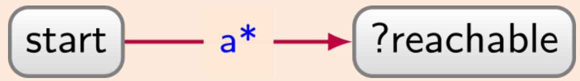
What else?



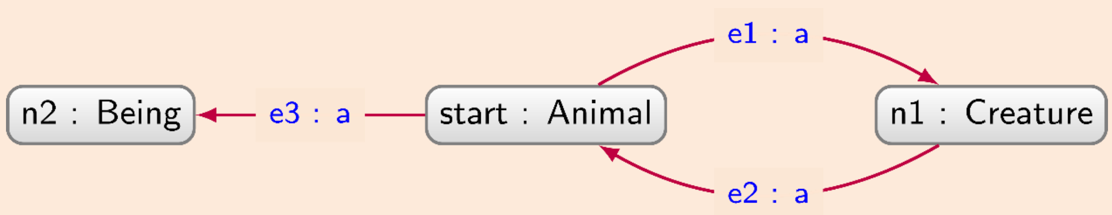
$$?p = \mathsf{TRAIL}\ \texttt{(start)=[a}^{*}\texttt{]=>(?reachable)}$$

No edge is repeated
in the path;
(We need property graphs)

What else?



$?p =$ TRAIL `(start)=[a*]=>(?reachable)`

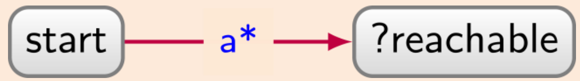


Good trails:

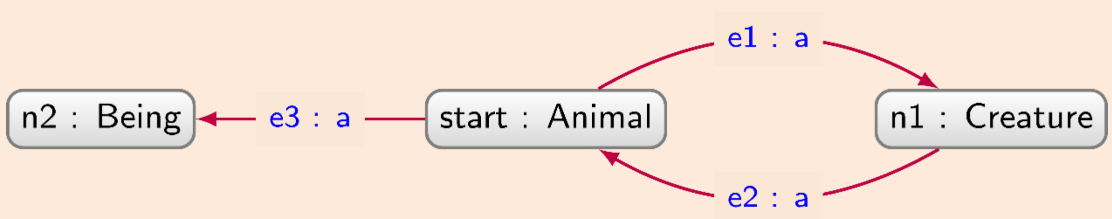
- start→n1
- start→n1→start
- start→n1→start→n2

(No infinite looping – limited by the number of edges)

What else?



$?p =$ TRAIL (start)=[a*]=>(?reachable)



Good trails:

- start→n1
- start→n1→start
- start→n1→start→n2

Not TRAIL

(No infinite looping – limited by the number of edges)

ALL OPTIONS

$$?p = \text{ANY WALK } \texttt{(start)=[regex]=>(?reachable)}$$

$$?p = \text{ANY SHORTEST WALK } \texttt{(start)=[regex]=>(?reachable)}$$

$$?p = \text{ALL SHORTEST WALK } \texttt{(start)=[regex]=>(?reachable)}$$

$$?p = \text{ANY SIMPLE } \texttt{(start)=[regex]=>(?reachable)}$$

$$?p = \text{ANY SHORTEST SIMPLE } \texttt{(start)=[regex]=>(?reachable)}$$

$$?p = \text{ALL SHORTEST SIMPLE } \texttt{(start)=[regex]=>(?reachable)}$$

$$?p = \text{SIMPLE } \texttt{(start)=[regex]=>(?reachable)}$$

$$?p = \text{ANY TRAIL } \texttt{(start)=[regex]=>(?reachable)}$$

...

ALL OPTIONS

Let's solve all these!!!

$?p =$ ANY WALK `(start)=[regex]=>(?reachable)`

$?p =$ ANY SHORTEST WALK `(start)=[regex]=>(?reachable)`

$?p =$ ALL SHORTEST WALK `(start)=[regex]=>(?reachable)`

$?p =$ ANY SIMPLE `(start)=[regex]=>(?reachable)`

$?p =$ ANY SHORTEST SIMPLE `(start)=[regex]=>(?reachable)`

$?p =$ ALL SHORTEST SIMPLE `(start)=[regex]=>(?reachable)`

$?p =$ SIMPLE `(start)=[regex]=>(?reachable)`

$?p =$ ANY TRAIL `(start)=[regex]=>(?reachable)`

...

ALL OPTIONS

PROVISO:
Starting node is fixed!

$?p =$ ANY WALK `(start)=[regex]=>(?reachable)`

$?p =$ ANY SHORTEST WALK `(start)=[regex]=>(?reachable)`

$?p =$ ALL SHORTEST WALK `(start)=[regex]=>(?reachable)`

$?p =$ ANY SIMPLE `(start)=[regex]=>(?reachable)`

$?p =$ ANY SHORTEST SIMPLE `(start)=[regex]=>(?reachable)`

$?p =$ ALL SHORTEST SIMPLE `(start)=[regex]=>(?reachable)`

$?p =$ SIMPLE `(start)=[regex]=>(?reachable)`

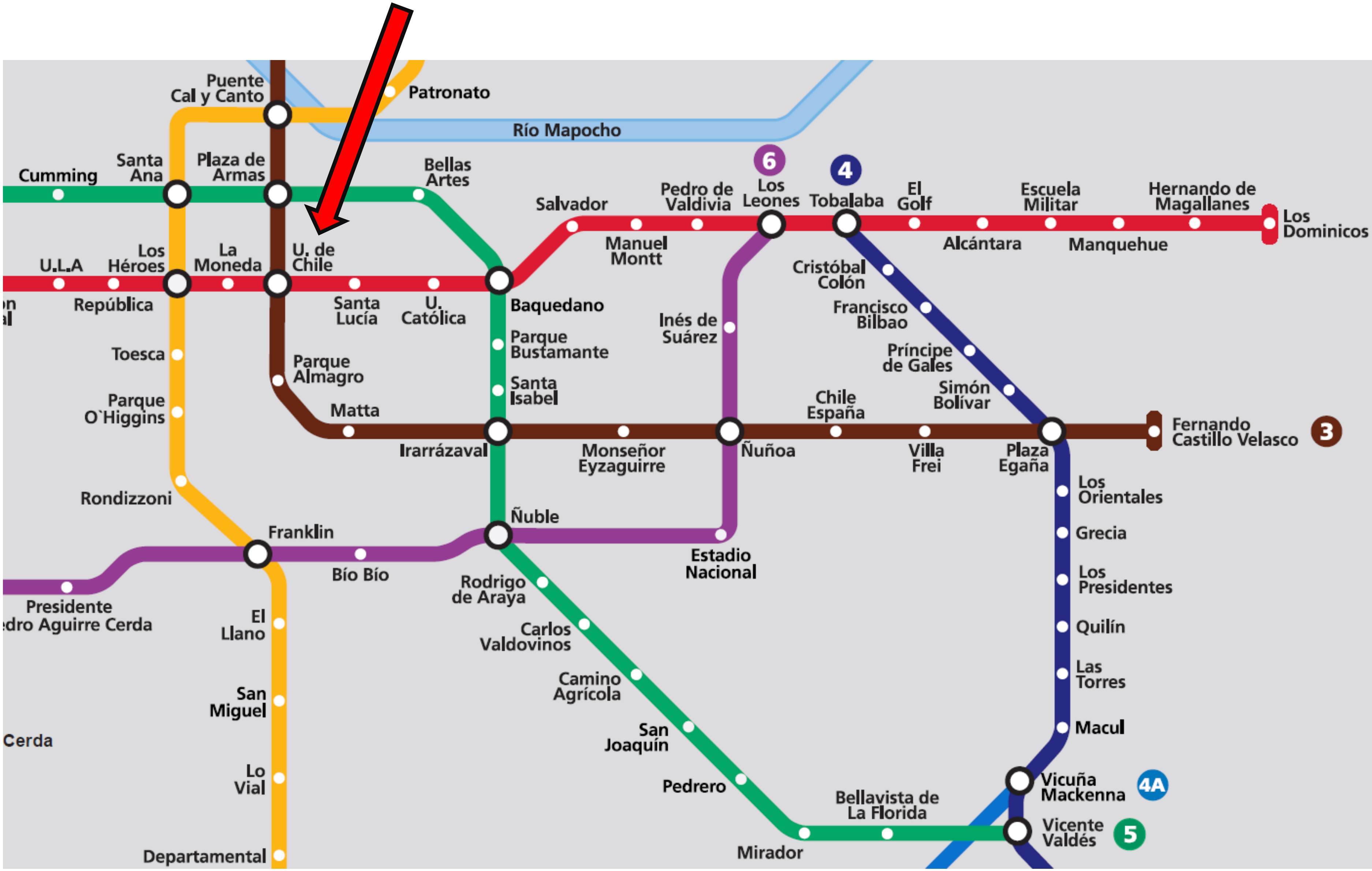
$?p =$ ANY TRAIL `(start)=[regex]=>(?reachable)`

...

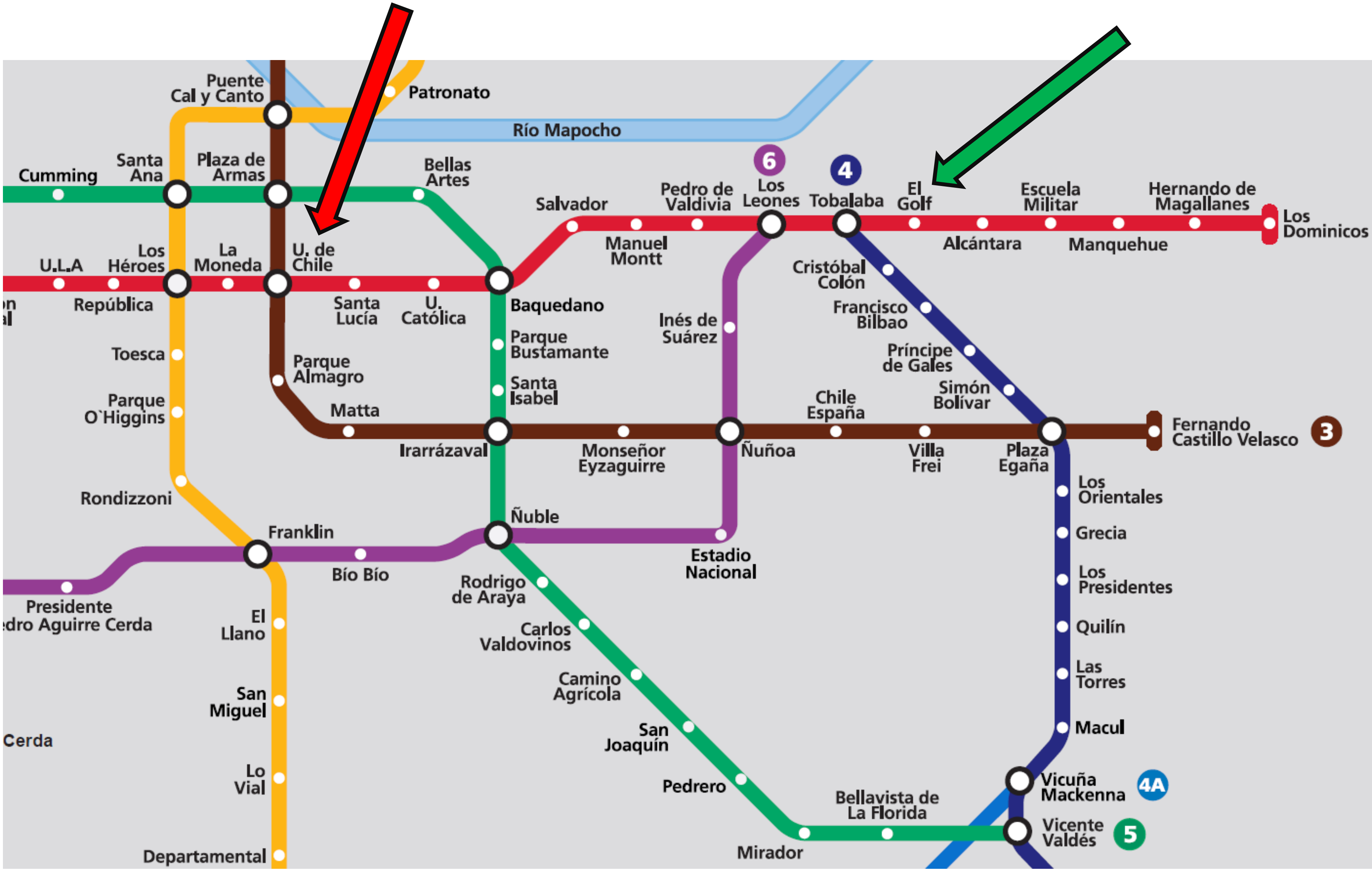
EXAMPLES

Puente Cal y Canto
Patronato
Río Mapocho
Cumming
Santa Ana
Plaza de Armas
Bellas Artes
Salvador
Pedro de Valdivia
Los Leones
Tobalaba
El Golf
Escuela Militar
Hernando de Magallanes
Los Dominicos
6
4
Manuel Montt
Alcántara
Manquehue
U.L.A
Los Héroes
La Moneda
U. de Chile
República
Santa Lucía
U. Católica
Baquedano
Cristóbal Colón
Francisco Bilbao
Príncipe de Gales
Simón Bolívar
Inés de Suárez
Toesca
Parque Bustamante
Parque Almagro
Santa Isabel
Parque O'Higgins
Matta
Chile España
Irarrázaval
Monseñor Eyzaguirre
Ñuñoa
Villa Frei
Plaza Egaña
Fernando Castillo Velasco
3
Rondizzoni
Los Orientales
Ñuble
Franklin
Grecia
Estadio Nacional
Bío Bío
Presidente
Rodrigo de Araya
Los Presidentes
El Llano
Carlos Valdovinos
Quilín
Camino Agrícola
Las Torres
San Miguel
San Joaquín
Macul
Cerda
Lo Vial
Pedrero
Vicuña Mackenna
4A
Bellavista de La Florida
Vicente Valdés
5
Departamental
Mirador

EXAMPLES


Puente
Cal y Canto
Patronato
Río Mapocho
Cumming
Santa
Ana
Plaza de
Armas
Bellas
Artes
6
4
Salvador
Pedro de
Valdivia
Los
Leones
Tobalaba
El
Golf
Escuela
Militar
Hernando de
Magallanes
Los
Dominicos
U.L.A
Los
Héroes
La
Moneda
U. de
Chile
Manuel
Montt
Alcántara
Manquehue
Cristóbal
Colón
República
Santa
Lucía
U.
Católica
Baquedano
Francisco
Bilbao
Inés de
Suárez
Parque
Bustamante
Príncipe
de Gales
Toesca
Parque
Almagro
Santa
Isabel
Simón
Bolívar
Parque
O`Higgins
Matta
Chile
España
Fernando
Castillo Velasco
3
Irarrázaval
Monseñor
Eyzaguirre
Ñuñoa
Villa
Frei
Plaza
Egaña
Los
Orientales
Rondizzoni
Ñuble
Franklin
Grecia
Estadio
Nacional
Bío Bío
Rodrigo
de Araya
Los
Presidentes
Presidente
Pedro Aguirre Cerda
El
Llano
Carlos
Valdovinos
Quilín
Camino
Agrícola
Las
Torres
San
Miguel
San
Joaquín
Macul
Cerda
Lo
Vial
Pedrero
Vicuña
Mackenna
4A
Bellavista de
La Florida
Vicente
Valdés
5
Departamental
Mirador

EXAMPLES



Puente Cal y Canto
Patronato
Río Mapocho
Cumming
Santa Ana
Plaza de Armas
Bellas Artes
6
Los Leones
4
Tobalaba
El Golf
Escuela Militar
Hernando de Magallanes
Los Dominicos
Salvador
Pedro de Valdivia
Manuel Montt
Alcántara
Manquehue
U.L.A
Los Héroes
La Moneda
U. de Chile
República
Santa Lucía
U. Católica
Baquedano
Cristóbal Colón
Francisco Bilbao
Príncipe de Gales
Simón Bolívar
Inés de Suárez
Toesca
Parque Bustamante
Parque Almagro
Santa Isabel
Parque O`Higgins
Matta
Irarrázaval
Monseñor Eyzaguirre
Ñuñoa
Chile España
Villa Frei
Plaza Egaña
Fernando Castillo Velasco
3
Rondizzoni
Los Orientales
Ñuble
Franklin
Grecia
Estadio Nacional
Bío Bío
Rodrigo de Araya
Presidente
Los Presidentes
El Llano
Carlos Valdovinos
Quilín
Camino Agrícola
Las Torres
San Miguel
San Joaquín
Macul
Cerda
Lo Vial
Pedrero
Vicuña Mackenna
4A
Bellavista de La Florida
Vicente Valdés
5
Departamental
Mirador

EXAMPLES

- Let us try out a few examples

https://mdb.imfd.cl/path_finder/

https://www.metro.cl/el-viaje/plano-de-red

Intermezzo

A bit of Theory

Instituto Milenio
Fundamentos
de los datos

What should theoreticians study?

PROBLEM:	Am I an answer?
INPUT:	Database D query q solution mapping μ
OUTPUT:	YES iff μ is in $q(D)$

- Usual approach: decision problems

What should theoreticians study?

PROBLEM:	Am I an answer?
INPUT:	Database D query q solution mapping μ
OUTPUT:	YES iff μ is in $q(D)$

- Does this make sense?
  - Join-eval is PTIME, but join + project NP-hard
- Algorithm for finding solutions:
  - Try all tuples one at a time

With graph databases this is even worse!

PROBLEM:	Am I an answer?
INPUT:	Graph database G path query q linking src to tgt path p from src to tgt
OUTPUT:	YES iff p is in $q(G)$

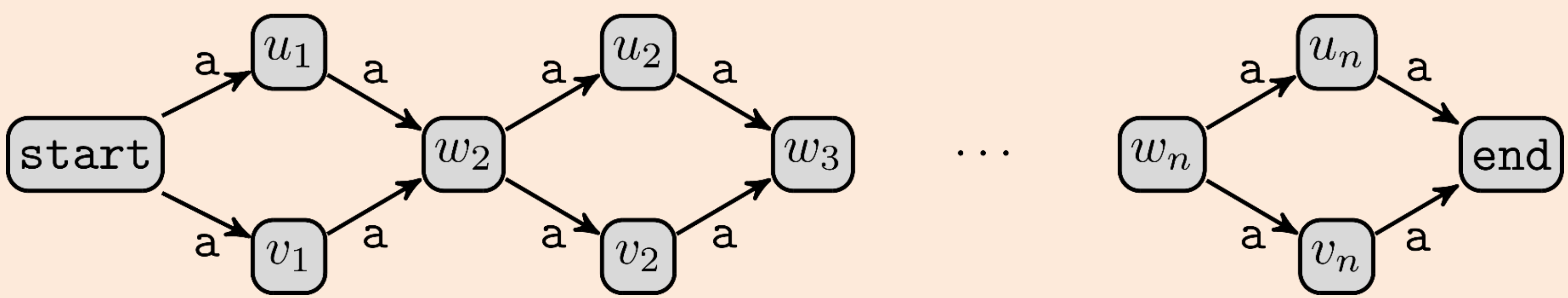
- For any reasonable notion of path query in PTIME

- How do we generate the results?
  - Iterate over all possible paths from *src* to *tgt*

Is this reasonable?

Sometimes there is an exponential number of those!

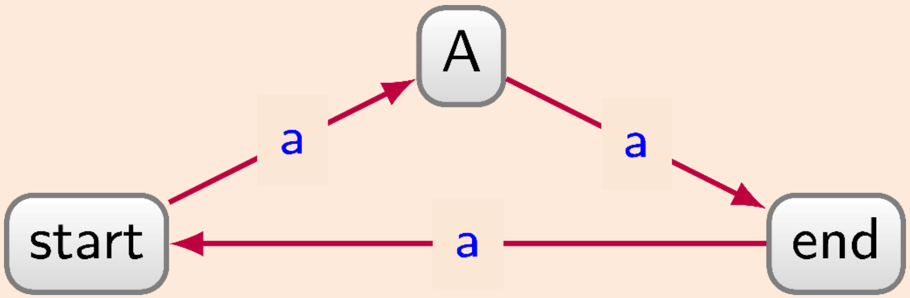
$$\texttt{query := } \; ?\mathrm{p} = \textsf{ALL SHORTEST WALK}\ (\texttt{start}) = [a^*] => (\texttt{end})$$



Is this reasonable?

Or infinite!

$\texttt{query} \; \texttt{:=} \quad ?\mathrm{p} = \mathsf{ALL\ PATHS}\ (\texttt{start}) = [a^*] \Rightarrow (\texttt{end})$



This is actually a semantic issue!

- start→A→end
- start→A→end→start→A→end
- start→A→end→start→A→end→start→A→end
- ...

Enumeration algorithms

What do I do when the output is exponential?

Measure the complexity in terms of |Input| + |Output|

Desiderata:

- Single pass over the data
- Enumerate results one by one without repetitions
- Ideally as soon as they are detected (pipelining)

Enumeration algorithms

What do I do when the output is exponential?

Enumeration algorithms:

- A pre-processing phase that „encodes” the outputs
- Enumeration phase that produces the results

Ideal case – constant delay:

- Single pass over the data O(|G|)
- Produce each output in time O(1)
- So complexity is **|Input|+|Output|**

Enumeration algorithms

What do I do when the output is exponential?

Can we produce a path in O(1)?

- n1→n2→n3→n4→n5→... ... ... →n_k

Graph/path case – output-linear delay:

- Single pass over the data O(|G|)
- Produce each output path p in time O(|p|)
  - We take O(1) for each element of the path we output
  - Basically the time needed to write down the path
- So complexity is **|Input|+|Output|**

Enumeration algorithms

These have been studied by the PODS community a lot!

Constant delay notion over relational

- Output is a single element per variable
- Usually **O(c·|Input|)** complexity with large c [Segoufin13]

Output-linear delay needed in general

- Used for RegEx analysis [REmatch]
- And very natural for path outputs

Enumeration algorithms

What do I want for graphs/paths?

Desiderata:

- Single pass over the data **O(|q|·|Input|)**
  - That can be done incrementally
  - Finding the first result pauses the algorithm
  - So the complexity will usually be proportional to path size
- Enumerate results one by one without repetitions
  - As soon as they are detected (pipelining)
  - With output-linear delay (even in the pipelined setting)

Let me show you how this was solved in '87

Any (shortest) walk

ANY WALK

ANY (SHORTEST)? WALK $(v) = [\texttt{regex}] \Rightarrow (?x)$

Theorem. *Let G be a graph database and q the query:*

$$\text{ANY (SHORTEST)? WALK } (v) = [\texttt{regex}] => (?x)$$

Computing the output of q over G can be done with $O(|\texttt{regex}| \times |G|)$ pre-processing and output-linear delay.

How?

Here is how

The product construction [MW95]:

- Graph is an automaton
- Regular expression is an automaton
- Do the cross product (on-the-fly to be "efficient")
- Do reachability check from start states to end states

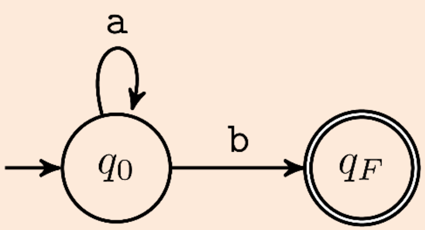
Which algorithms can do this?

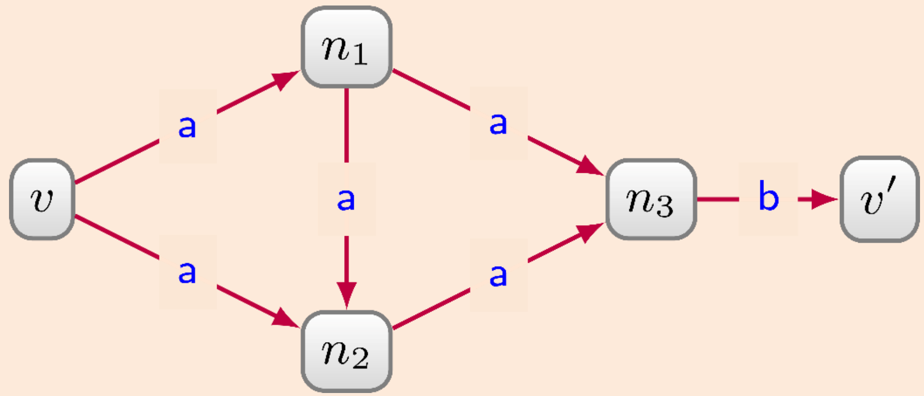
- BFS
- DFS
- A*
- IDDFS
- ...

See [MW95,BDRV17,B13,FMRV23] for details

Basic idea

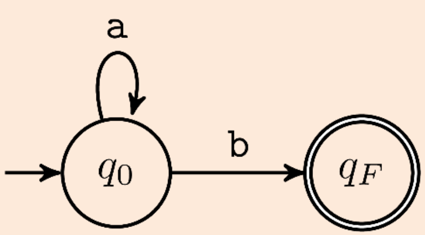
ANY WALK $(v) = [\texttt{a}^*\texttt{b}] \Rightarrow (?x)$

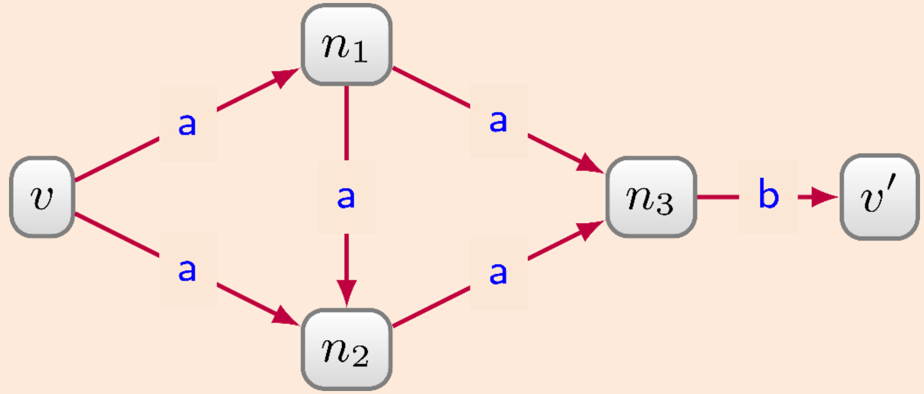




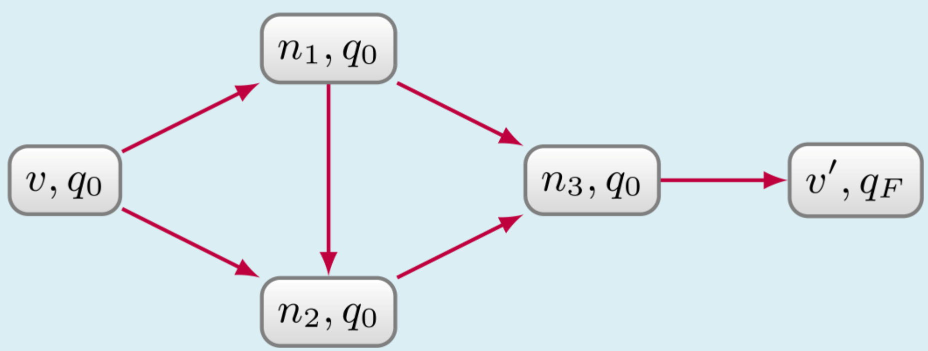
Basic idea

ANY WALK $(v) = [\texttt{a}^*\texttt{b}] => (?x)$



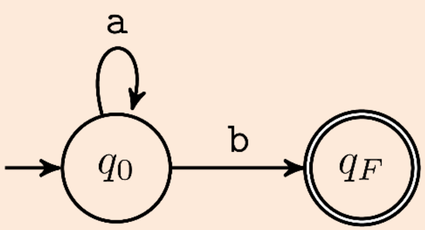


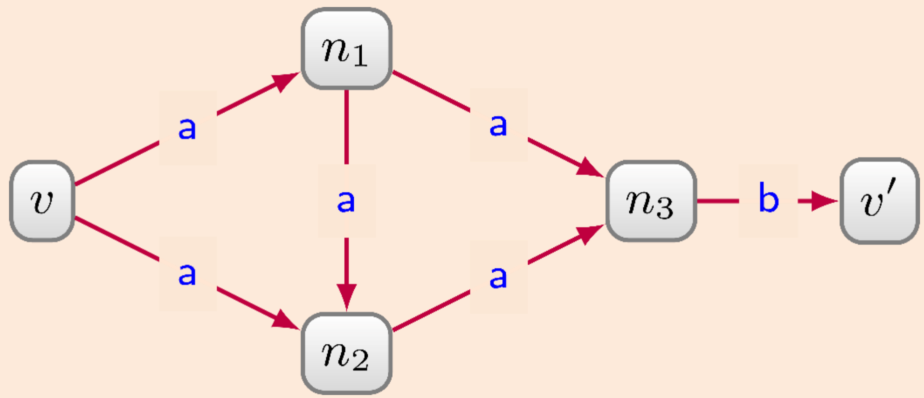
Product graph:



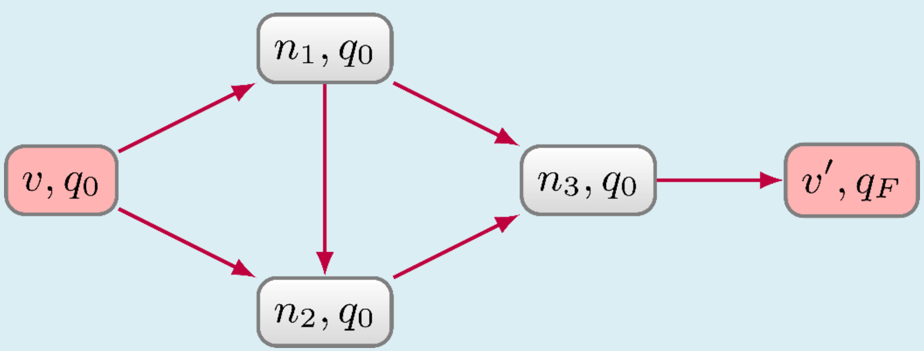
Basic idea

ANY WALK $(v) = [\mathtt{a}^*\mathtt{b}] => (?x)$



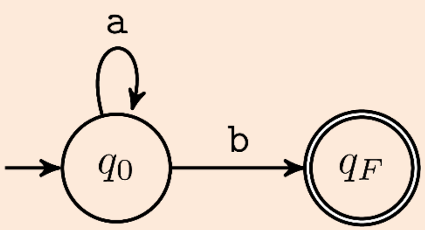


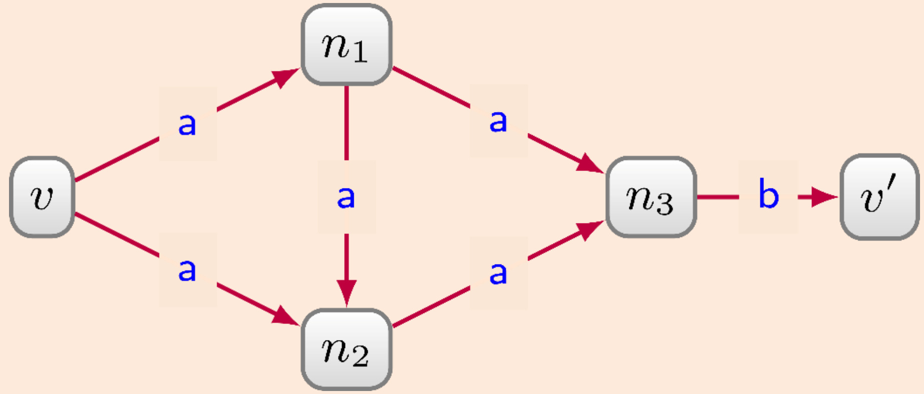
Product graph:



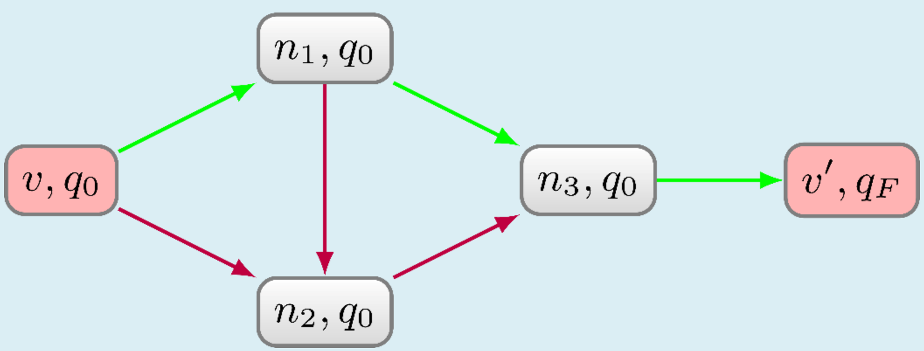
Basic idea

ANY WALK $(v) = [\texttt{a}^*\texttt{b}] \Rightarrow (?x)$





Product graph:



ANY WALK – on-the-fly

Algorithm 1 Algorithm for $?p$ = ANY WALK ($(v) = [$`regex`$] => (?x)$)

function $\textsc{AnyWalk}(G, q)$
  $\mathcal{A} \leftarrow Automaton($`regex`$)$ ▷ q_0 init; q_F final
  Open.init() ▷ Queue/Stack
  Visited.init() ▷ Dictionary
  start $\leftarrow (v, q_0, \perp)$
  Open.push(start)
  Visited.push(start)
  while !Open.isEmpty() **do**
    curr=Open.pop() ▷ $curr = (n, q, prev)$
    if $q == q_F$ **then** ▷ A solution is found
      getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
      if !(next $\in$ Visited) **then**
        next $= (n', q',$ curr$)$
        Open.push(next)
        Visited.push(next)

Let's see

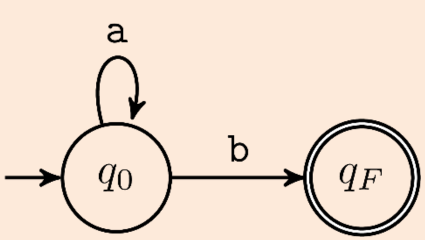
start $\leftarrow (v, q_0, \bot)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q',$ curr$)$
            Open.push(next)
            Visited.push(next)

ANY WALK $(v) = [\mathtt{a}^*\mathtt{b}] => (?x)$

Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

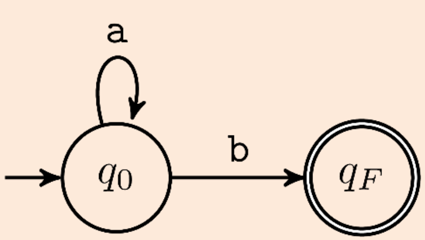
ANY WALK $(v) = [\mathtt{a}^*\mathtt{b}] => (?x)$

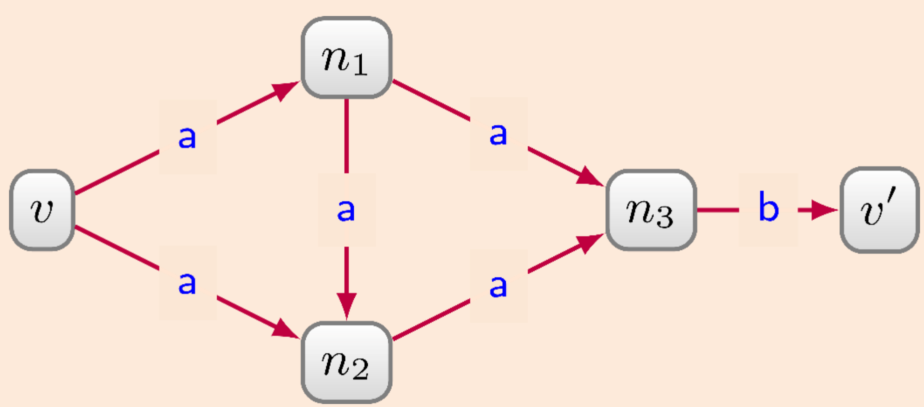


Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q',$ curr$)$
            Open.push(next)
            Visited.push(next)

ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$





Let's see

start $\leftarrow (v, q_0, \bot)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

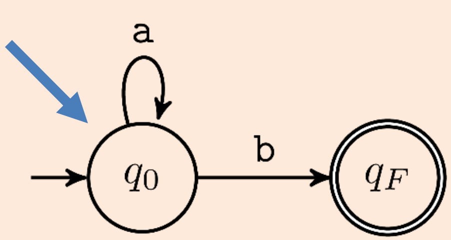
Open:

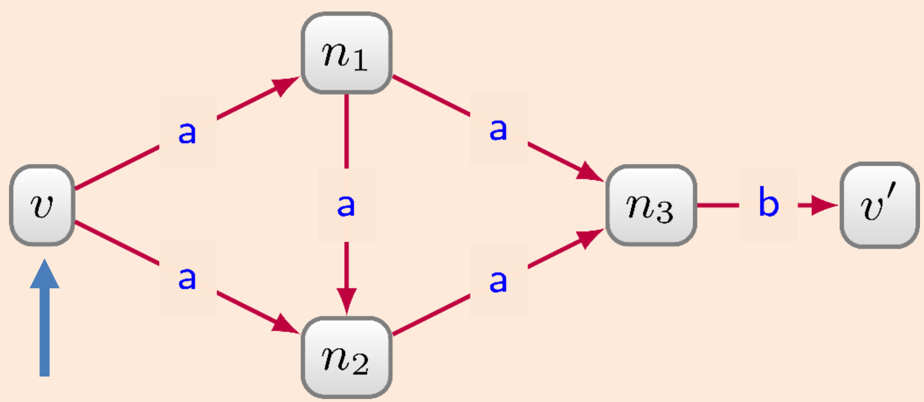
$(v, q_0, \bot)$

Visited:

(v, q_0)

ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$





Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

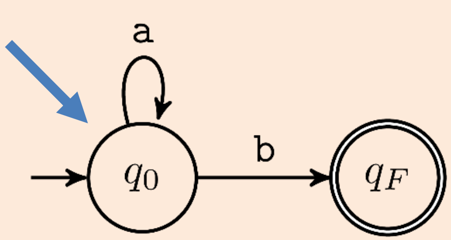
Open:

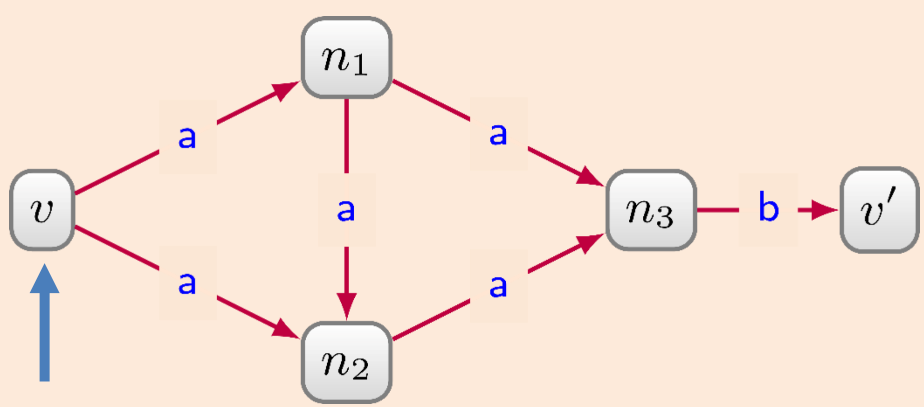
$(v, q_0, \perp)$

Visited:

(v, q_0)

ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$

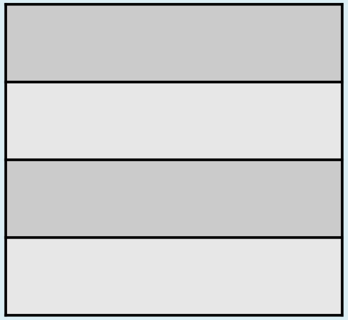




Let's see

start $\leftarrow (v, q_0, \bot)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

Open:

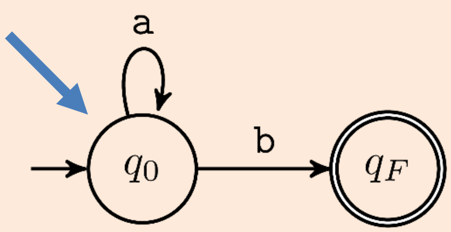


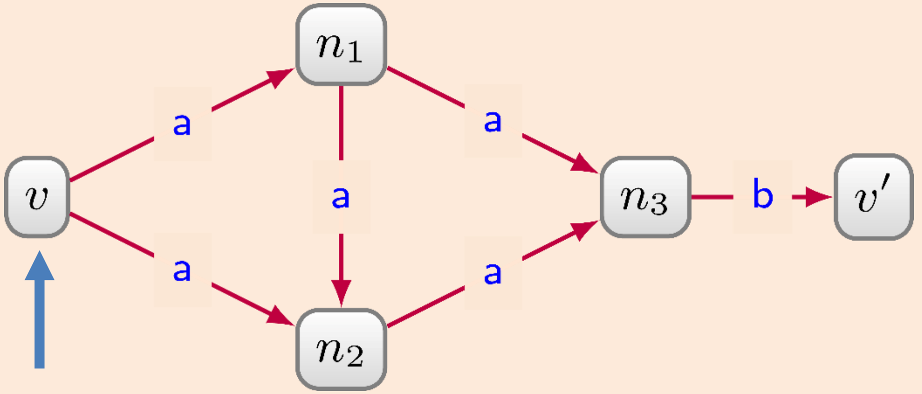
Visited:

(v, q_0)

curr

ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$

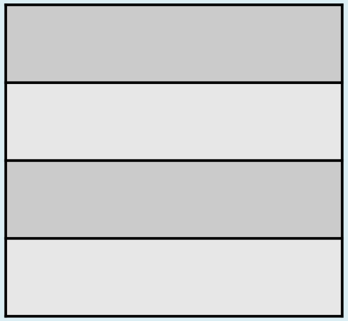




Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q',$ curr$)$
            Open.push(next)
            Visited.push(next)

Open:

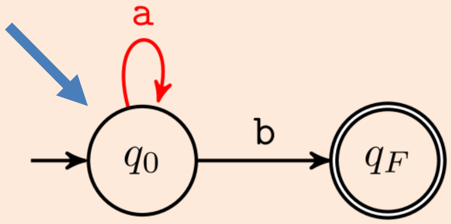


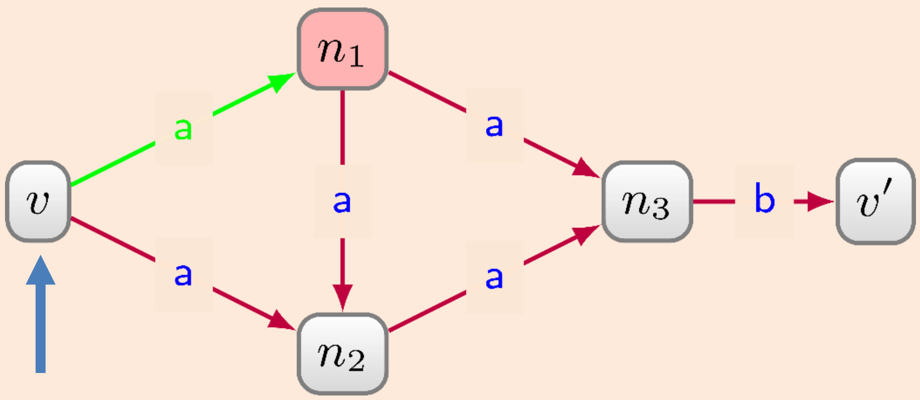
Visited:

(v, q_0)

curr

ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$

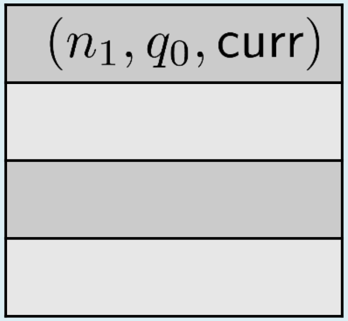




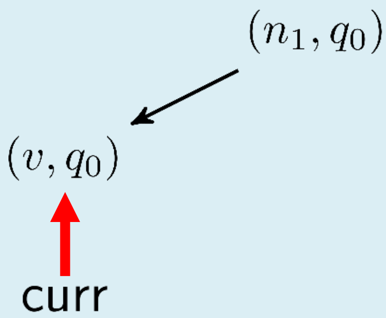
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

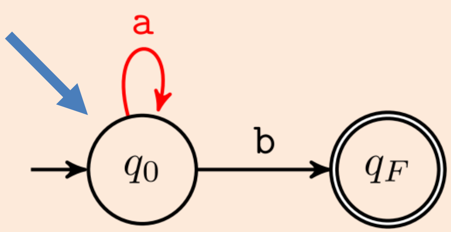
Open:

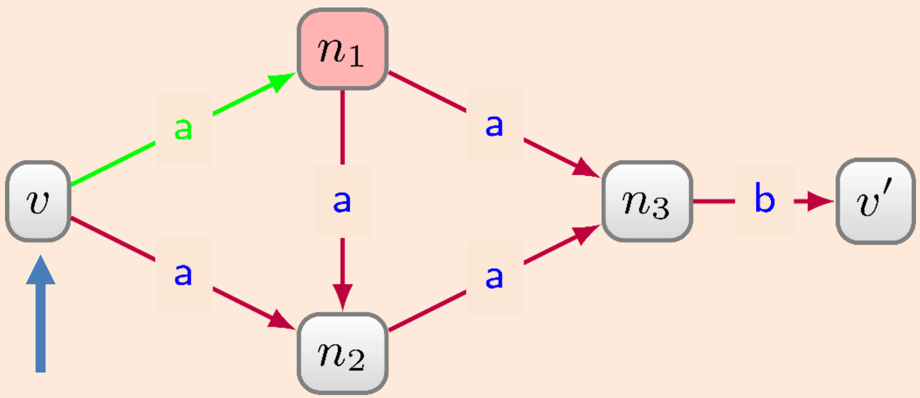


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$

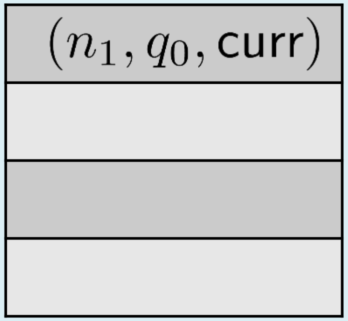




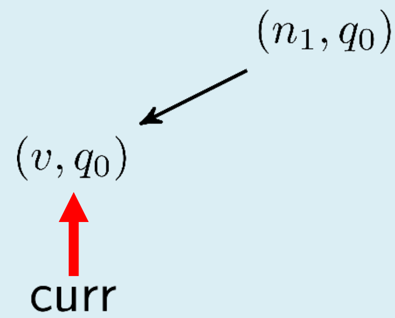
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

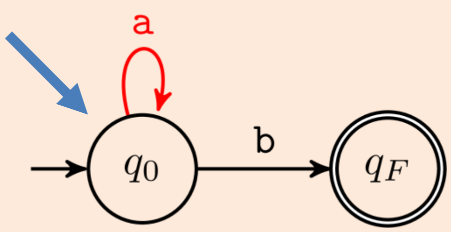
Open:

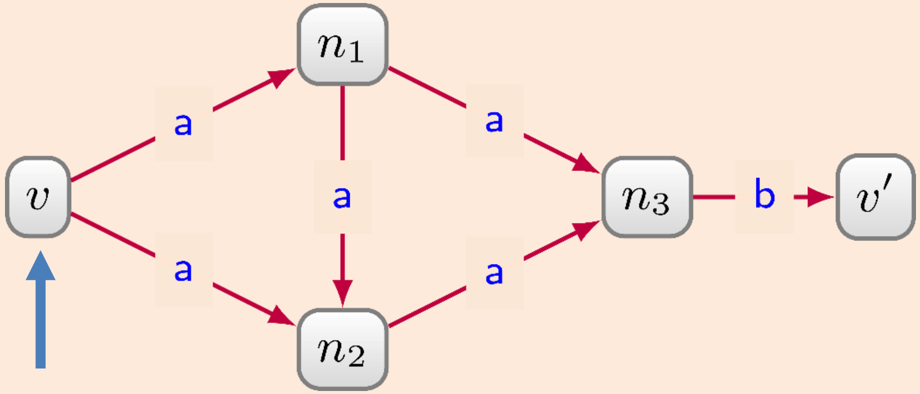


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$





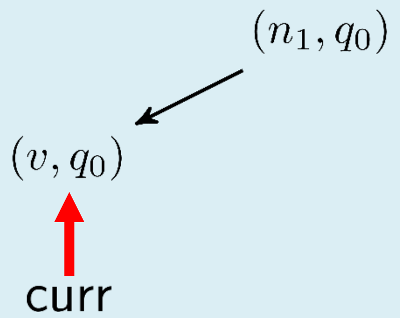
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

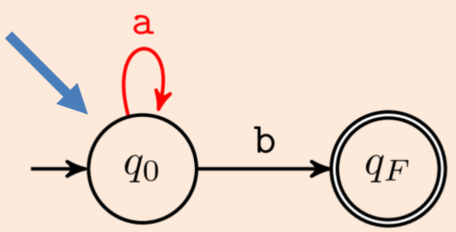
Open:

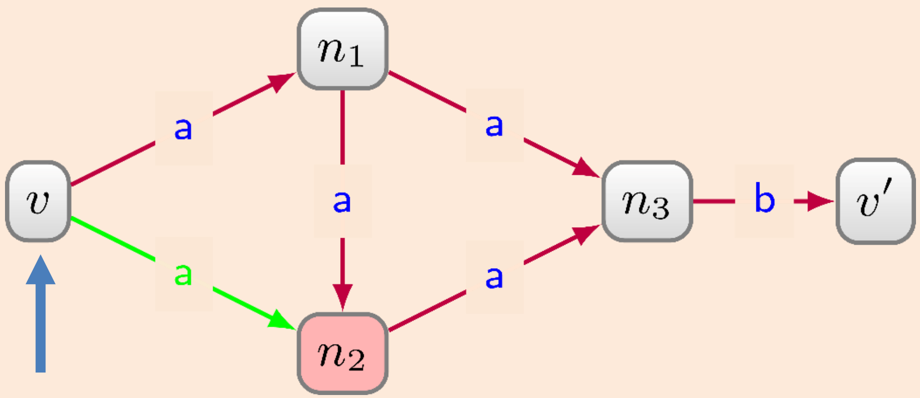
(n_1, q_0, curr)

Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$





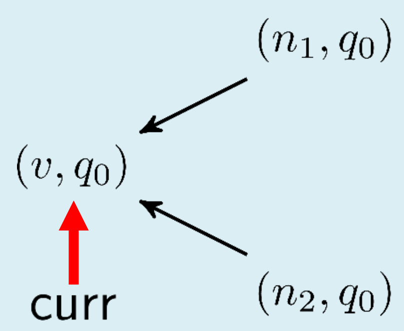
Let's see

start $\leftarrow (v, q_0, \bot)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    if !(next $\in$ Visited) **then**
      next $= (n', q', \text{curr})$
      Open.push(next)
      Visited.push(next)

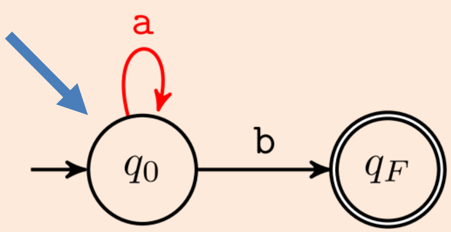
Open:

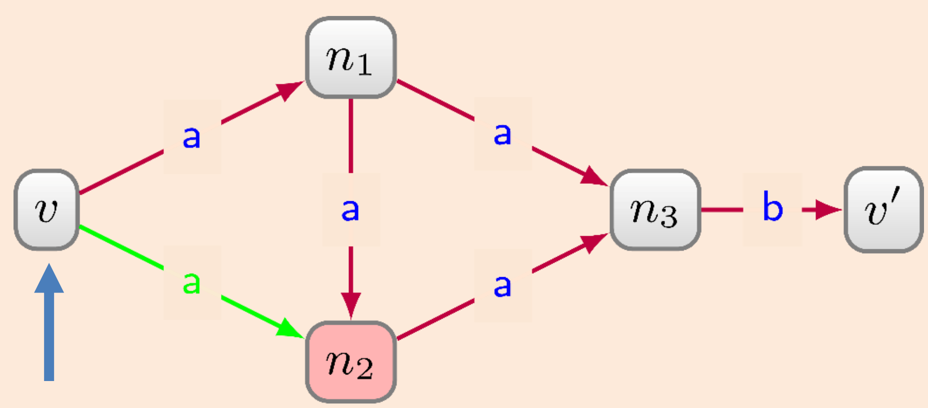
(n_1, q_0, curr)
(n_2, q_0, curr)

Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$





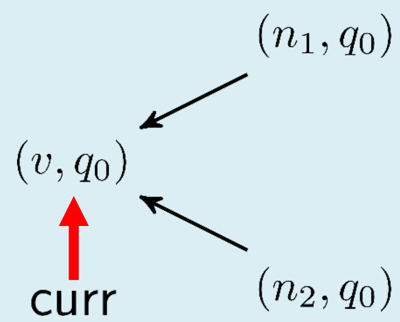
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    if !(next $\in$ Visited) **then**
      next $= (n', q',$ curr$)$
      Open.push(next)
      Visited.push(next)

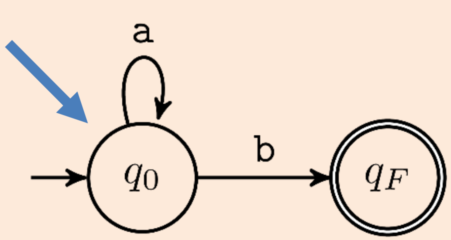
Open:

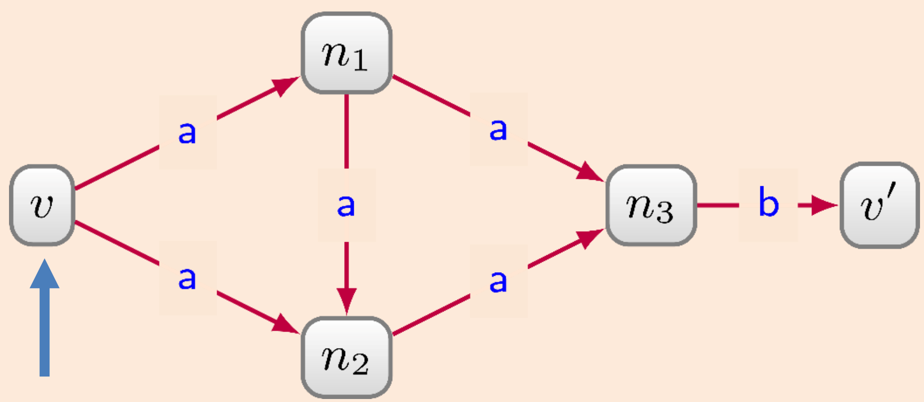
$(n_1, q_0,$ curr$)$
$(n_2, q_0,$ curr$)$

Visited:



ANY WALK $(v) = [$a*b$] => (?x)$

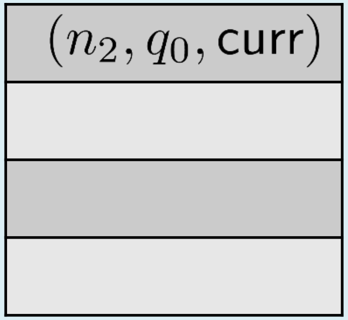


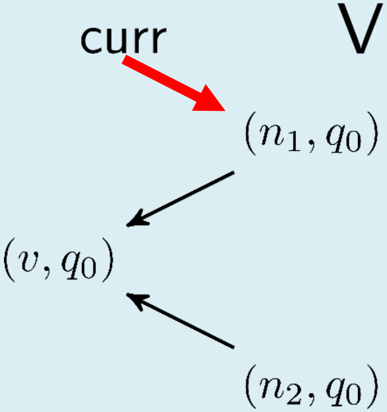


Let’s see

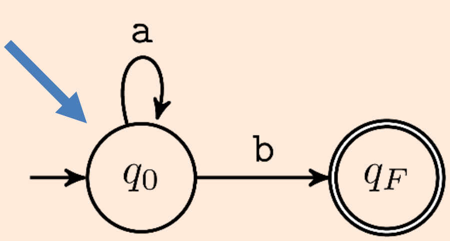
start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    if !(next $\in$ Visited) **then**
      next $= (n', q',$ curr$)$
      Open.push(next)
      Visited.push(next)

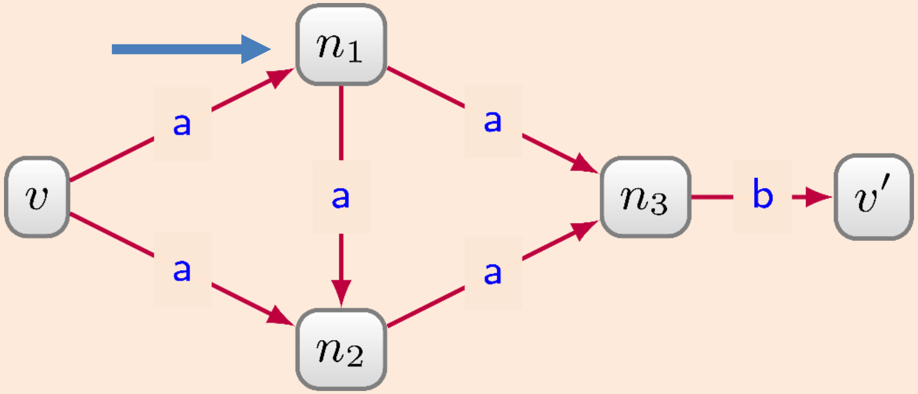
Open:





ANY WALK $(v) = [$a*b$] => (?x)$

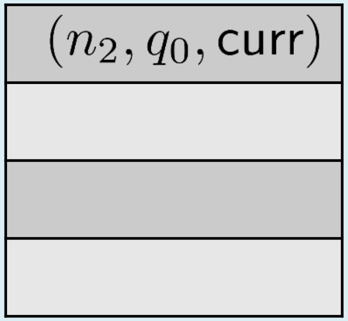




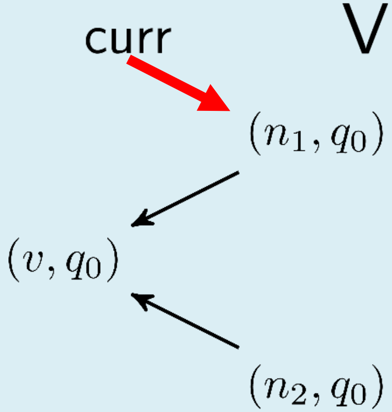
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    if !(next $\in$ Visited) **then**
      next $= (n', q', \text{curr})$
      Open.push(next)
      Visited.push(next)

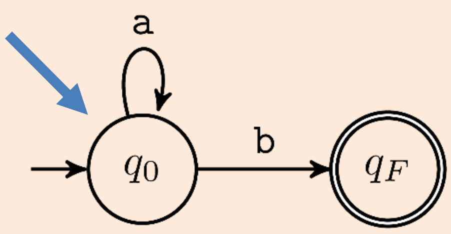
Open:

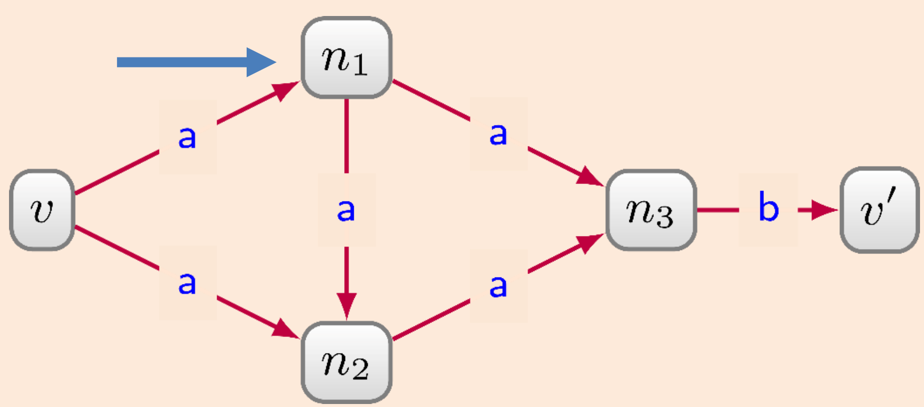


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$

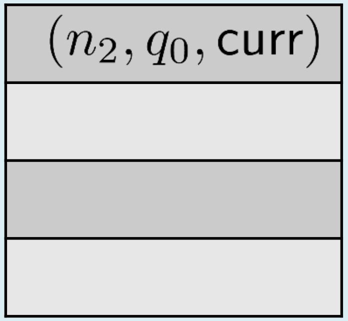


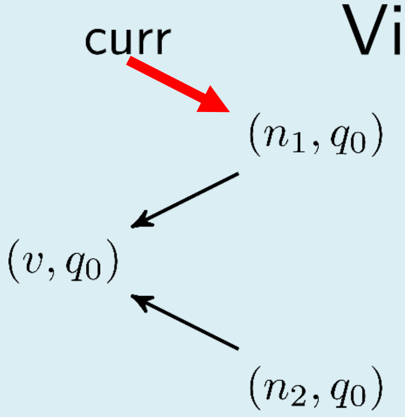


Let's see

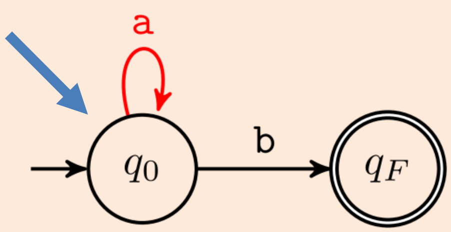
start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q',$ curr$)$
            Open.push(next)
            Visited.push(next)

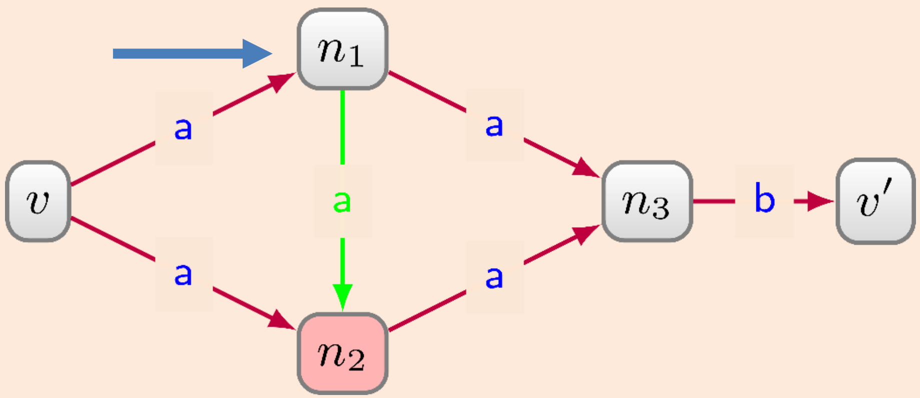
Open:





ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$

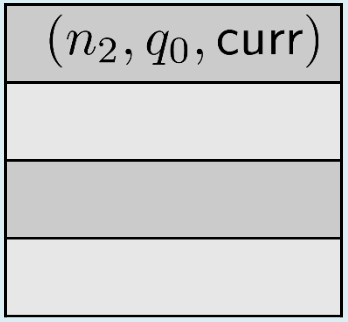




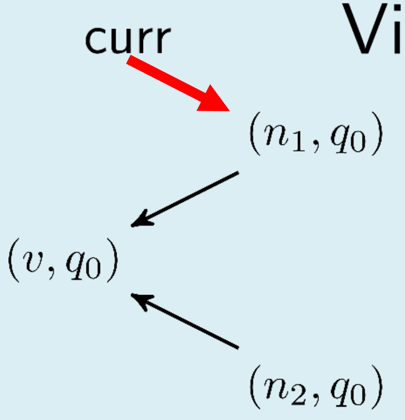
Let's see

```
start ← (v, q0, ⊥)
Open.push(start)
Visited.push(start)
while !Open.isEmpty() do
    curr=Open.pop()
    if q == qF then
        getPath(curr)
    for next = (n′, q′) ∈ Neighbours(curr) do
        if !(next ∈ Visited) then
            next = (n′, q′, curr)
            Open.push(next)
            Visited.push(next)
```

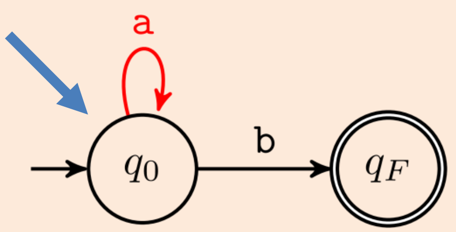
Open:

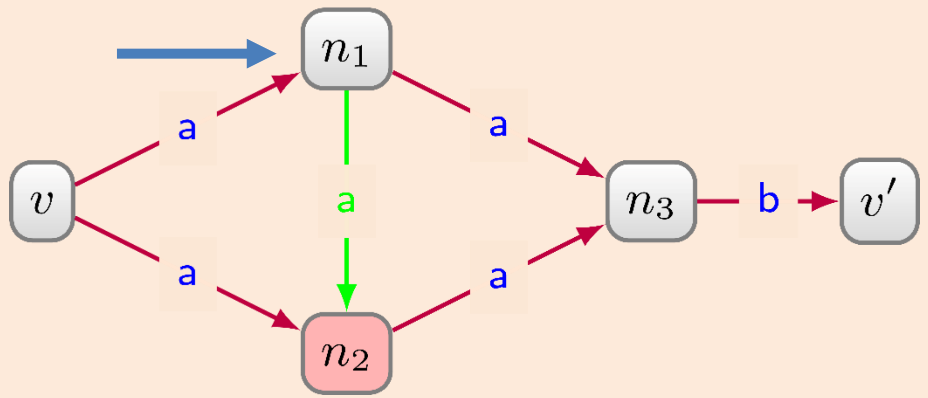


Visited:



ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$

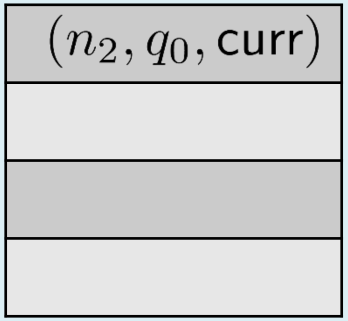




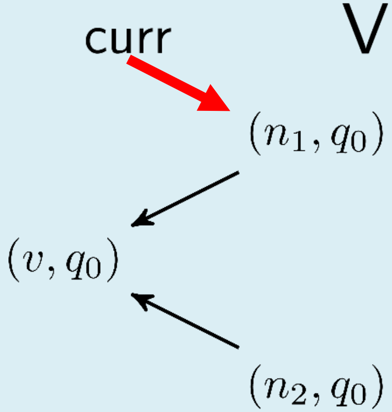
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

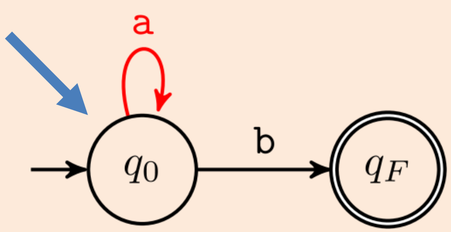
Open:

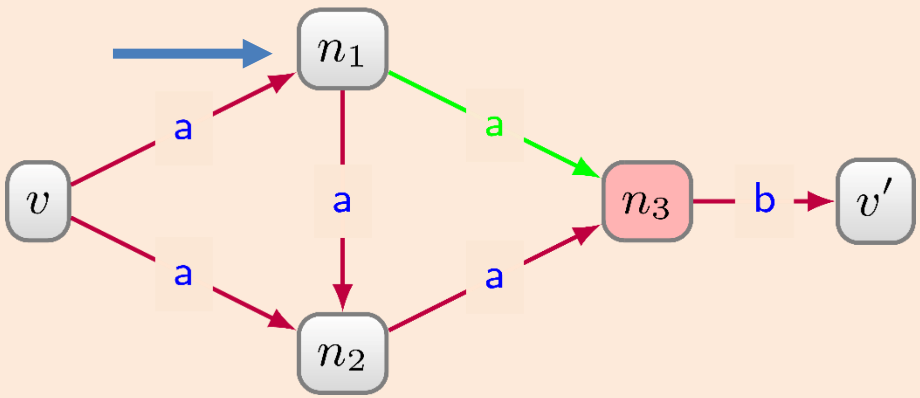


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$





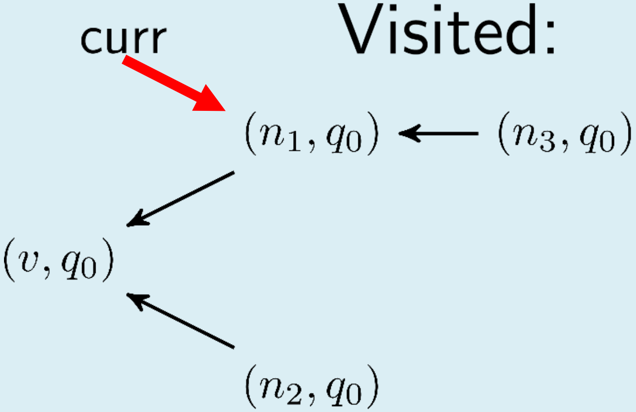
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q',$ curr$)$
            Open.push(next)
            Visited.push(next)

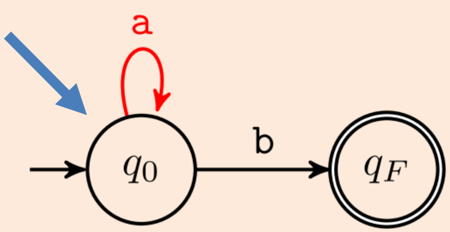
Open:

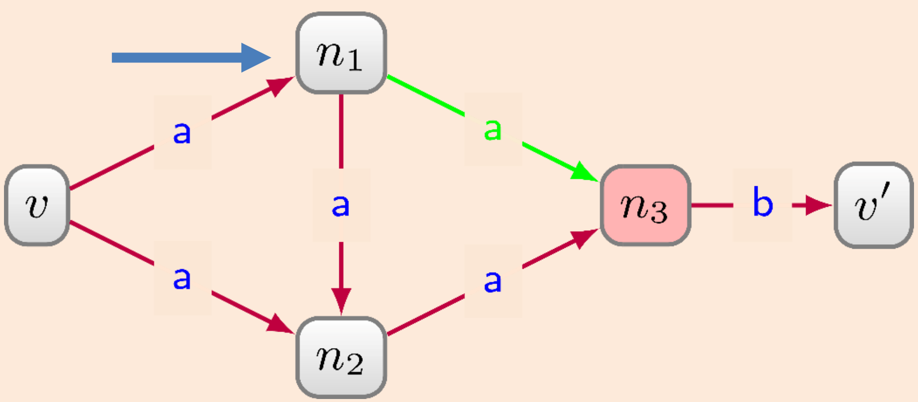
$(n_2, q_0,$ curr$)$
$(n_3, q_0,$ curr$)$

Visited:



ANY WALK $(v) = [$a*b$] => (?x)$





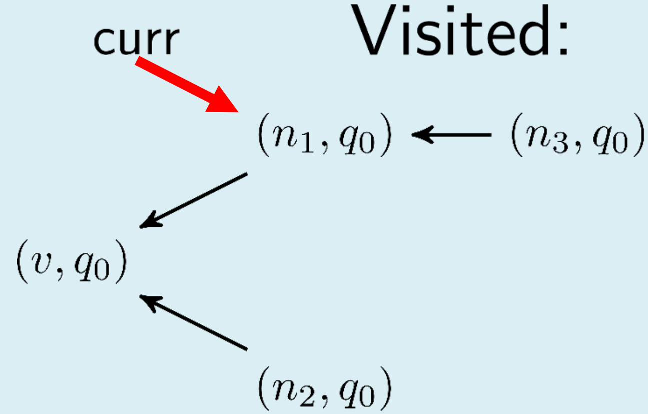
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q',$ curr$)$
            Open.push(next)
            Visited.push(next)

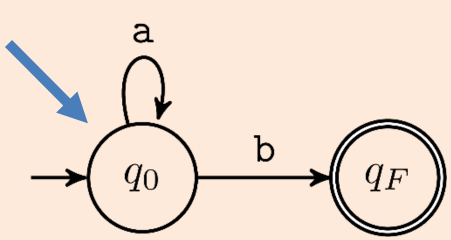
Open:

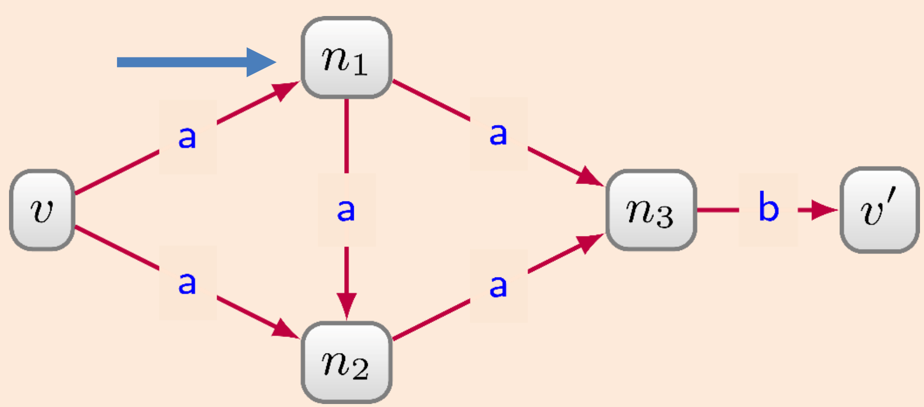
$(n_2, q_0,$ curr$)$
$(n_3, q_0,$ curr$)$

Visited:



ANY WALK $(v) =$ [a*b] $=> (?x)$

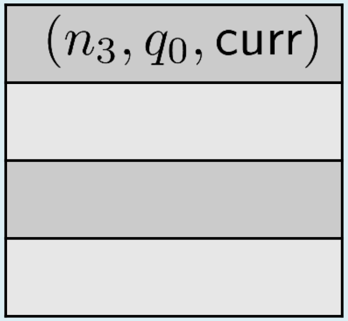




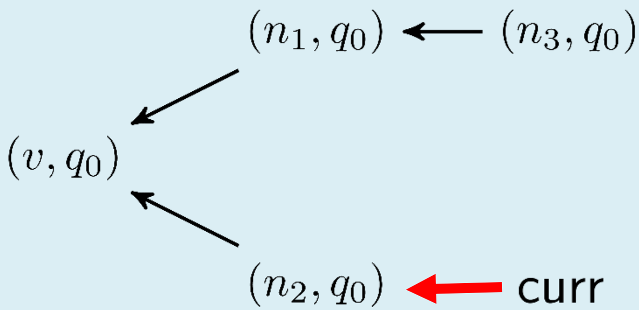
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

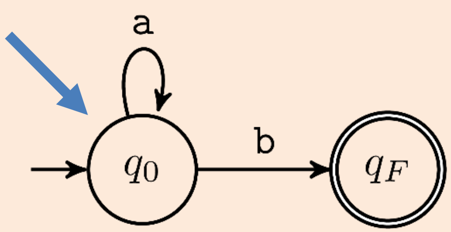
Open:

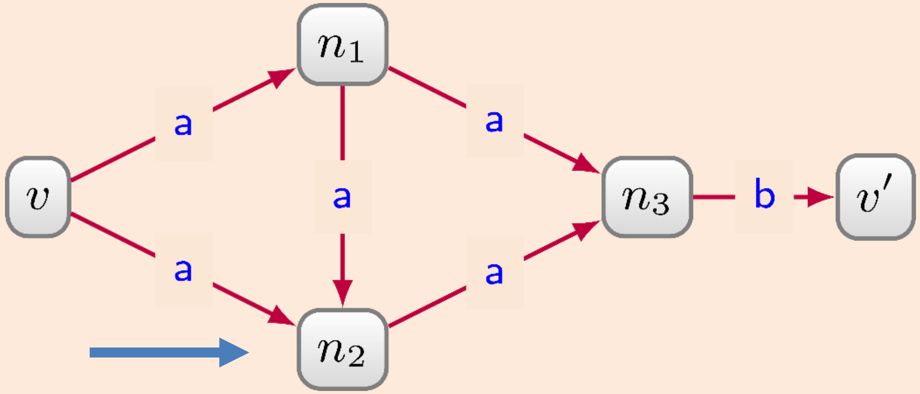


Visited:



ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$

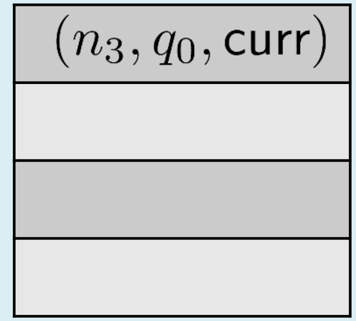




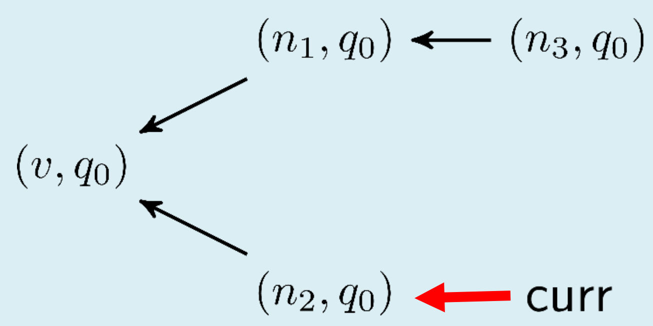
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

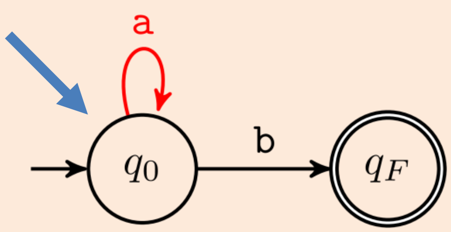
Open:

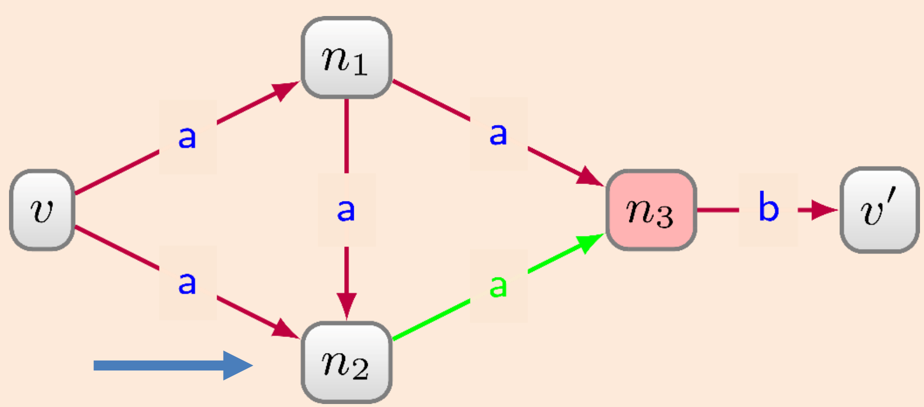


Visited:



ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$

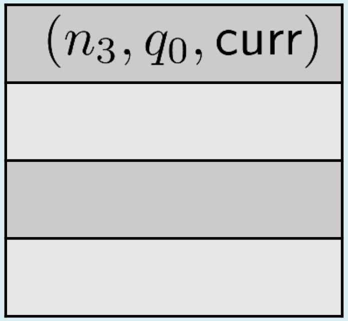




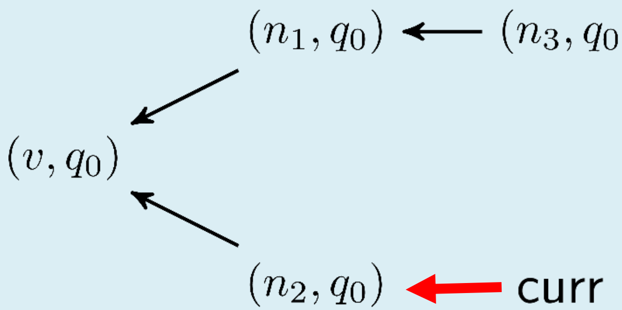
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    $\boxed{\textbf{if } !(\text{next} \in \text{Visited}) \textbf{ then}}$
      next $= (n', q', \text{curr})$
      Open.push(next)
      Visited.push(next)

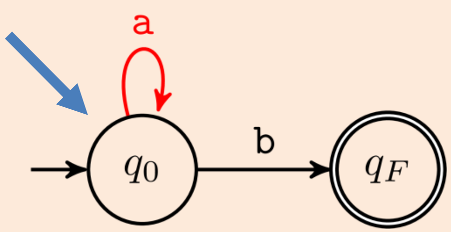
Open:

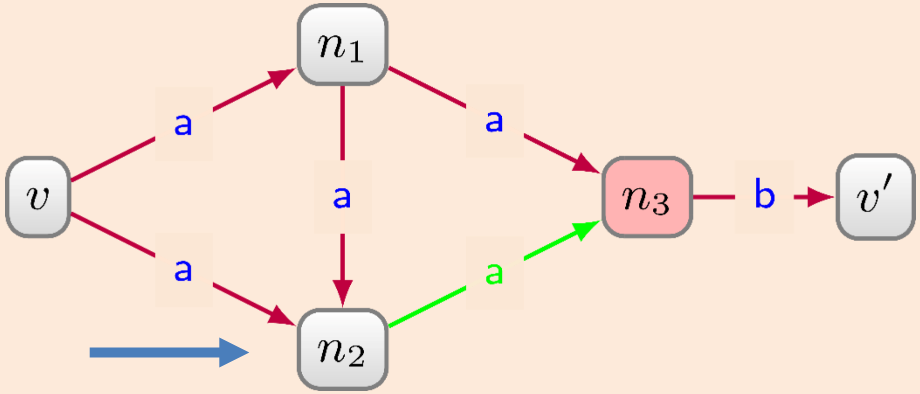


Visited:



ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$

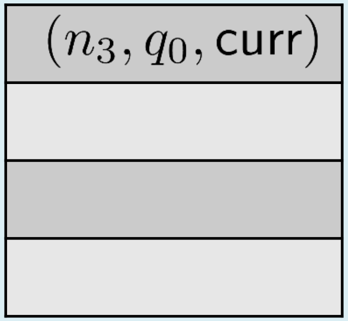




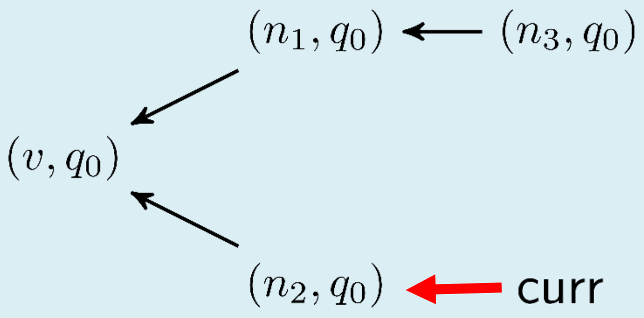
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    if !(next $\in$ Visited) **then**
      next $= (n', q', \text{curr})$
      Open.push(next)
      Visited.push(next)

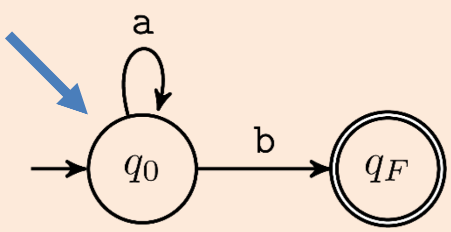
Open:

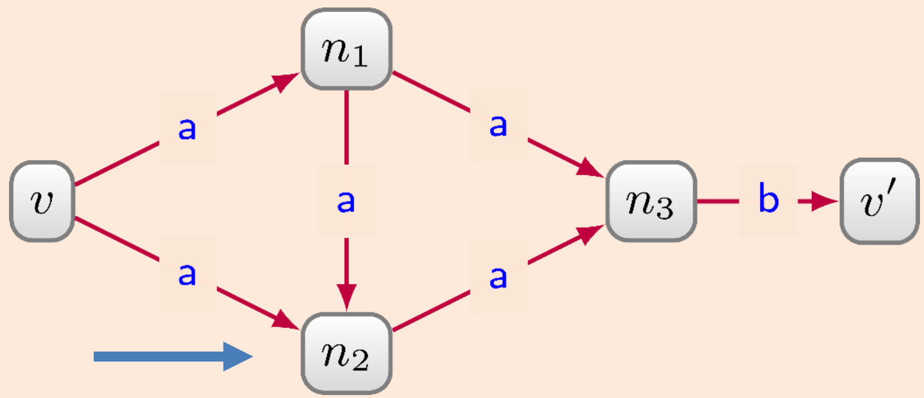


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$

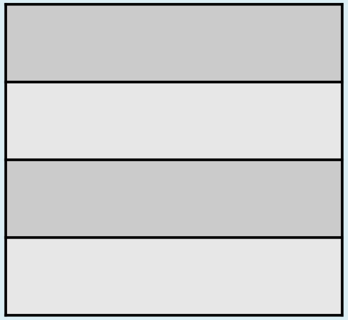




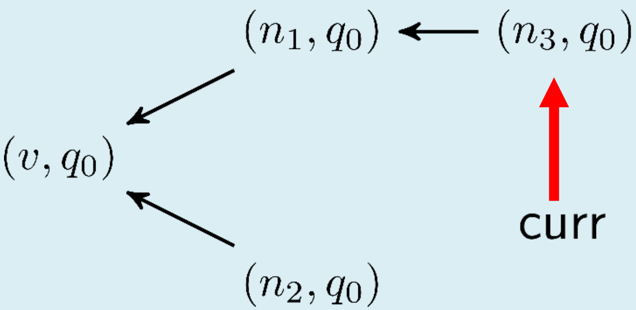
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    if !(next $\in$ Visited) **then**
      next $= (n', q',$ curr$)$
      Open.push(next)
      Visited.push(next)

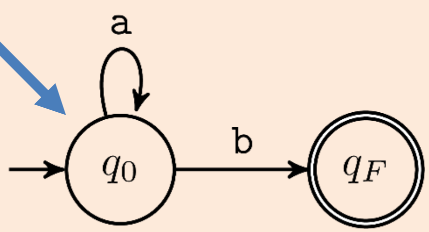
Open:

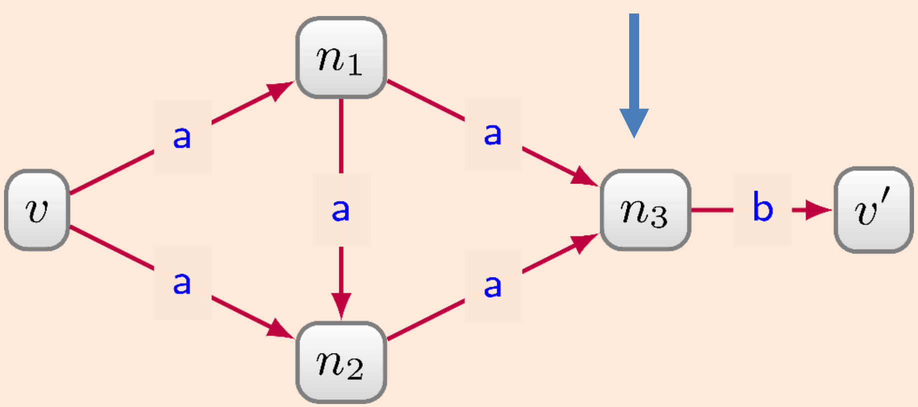


Visited:



ANY WALK $(v) =$ [a*b] $=> (?x)$

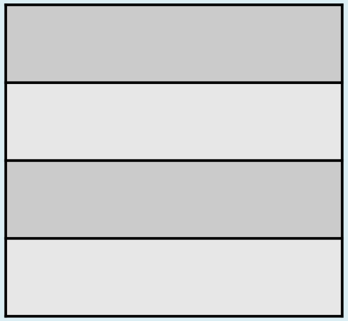




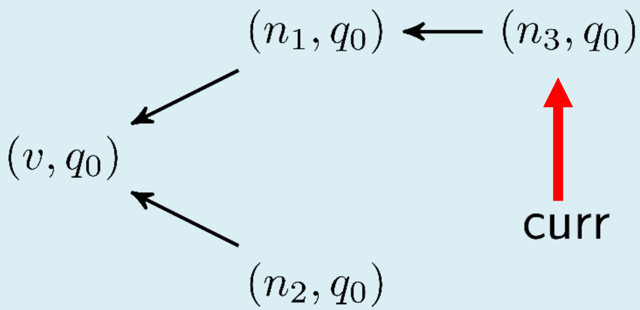
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    if !(next $\in$ Visited) **then**
      next $= (n', q', \text{curr})$
      Open.push(next)
      Visited.push(next)

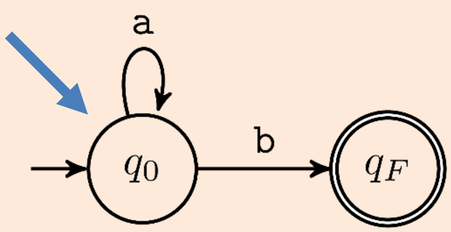
Open:

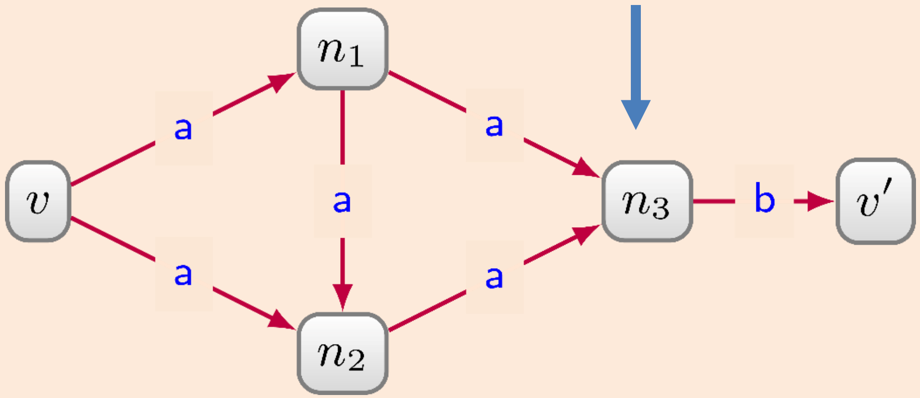


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$

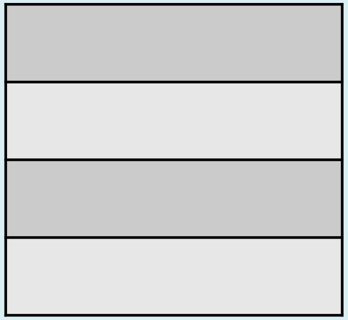




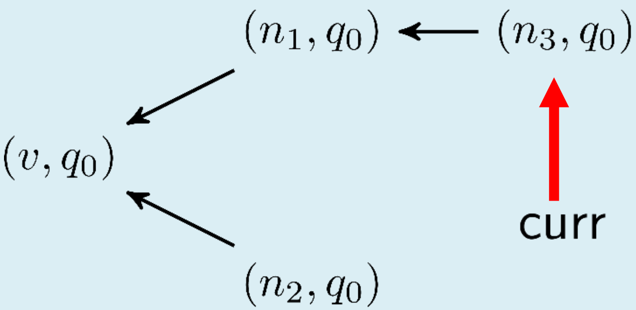
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

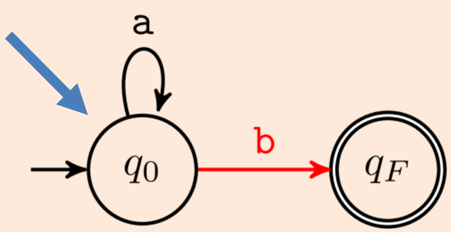
Open:

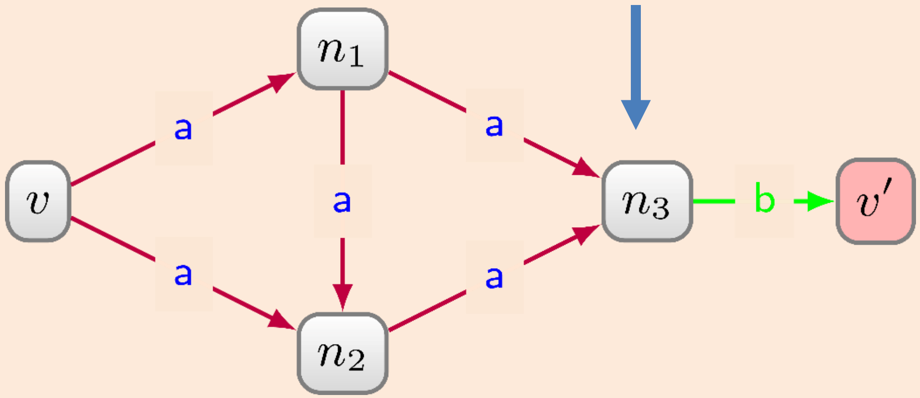


Visited:



ANY WALK $(v) = [\mathsf{a}^*\mathsf{b}] => (?x)$

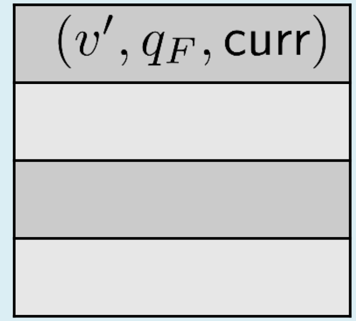




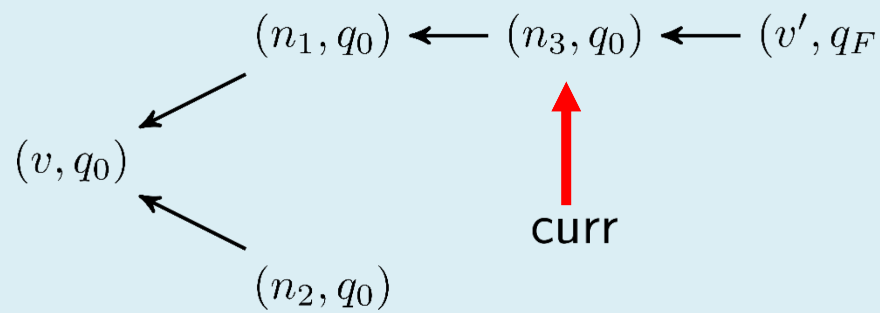
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr$=$Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

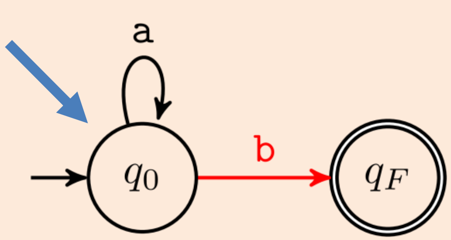
Open:

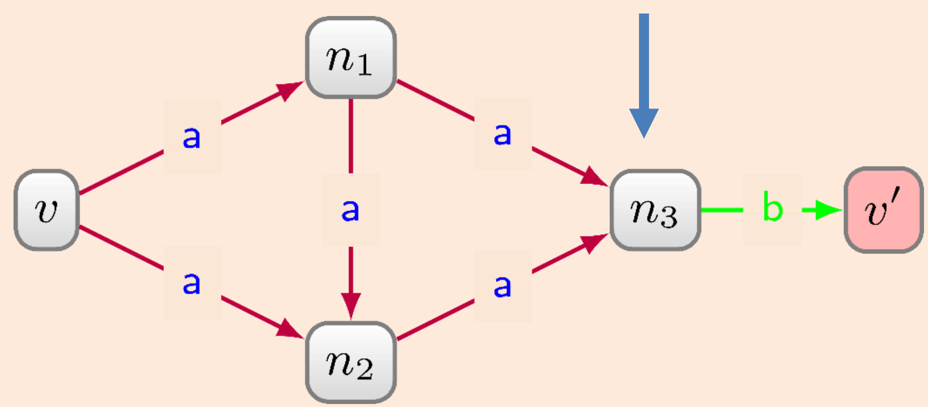


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$

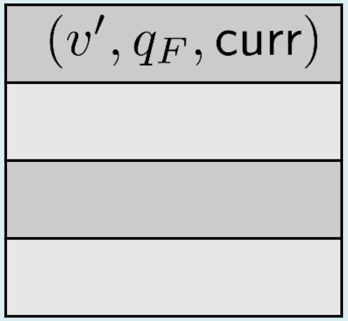




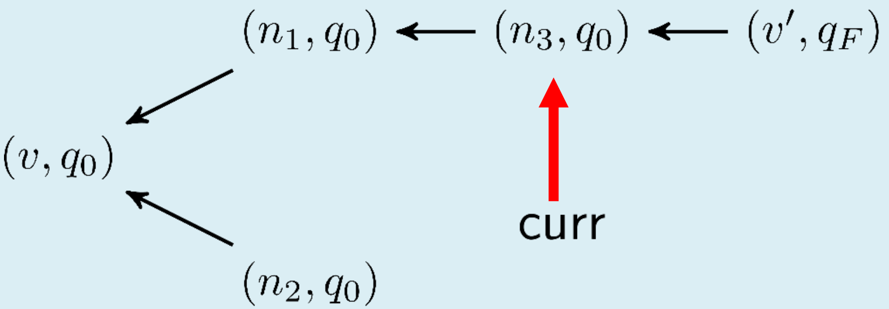
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q', \text{curr})$
            Open.push(next)
            Visited.push(next)

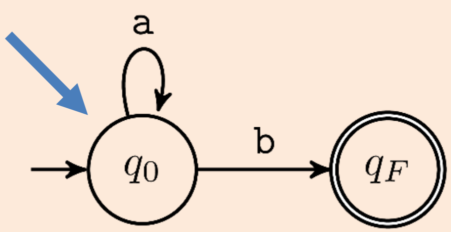
Open:

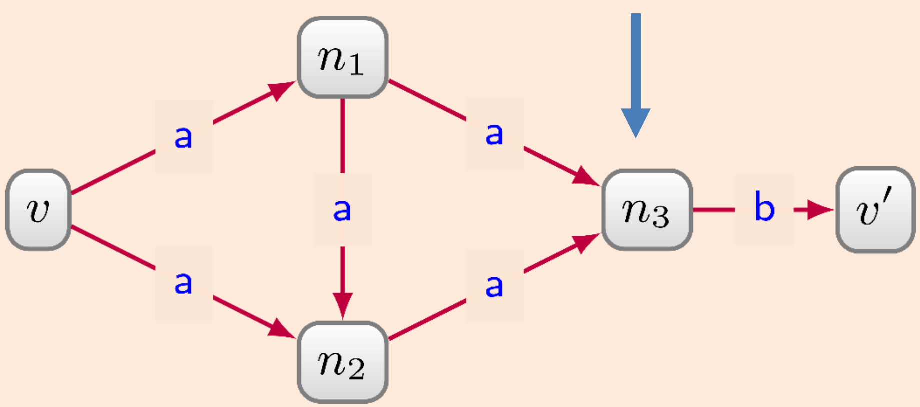


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$

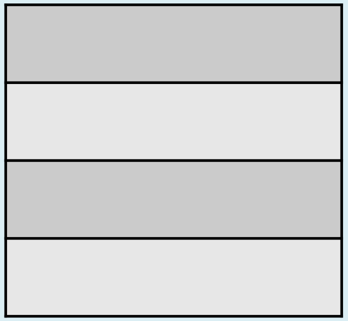




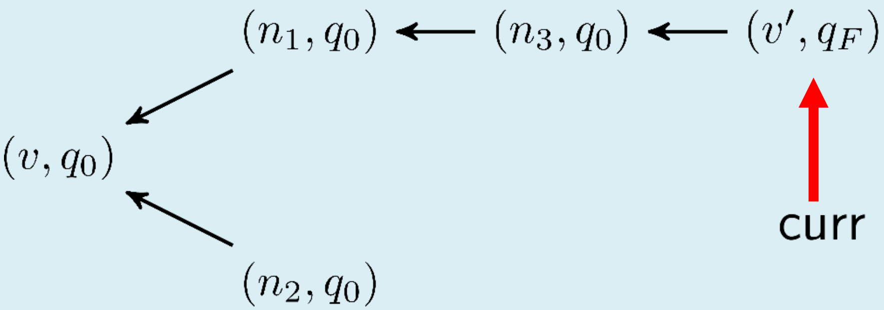
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    curr=Open.pop()
    if $q == q_F$ **then**
        getPath(curr)
    for next $= (n', q') \in$ Neighbours(curr) **do**
        if !(next $\in$ Visited) **then**
            next $= (n', q',$ curr$)$
            Open.push(next)
            Visited.push(next)

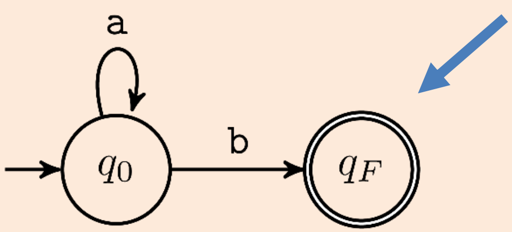
Open:

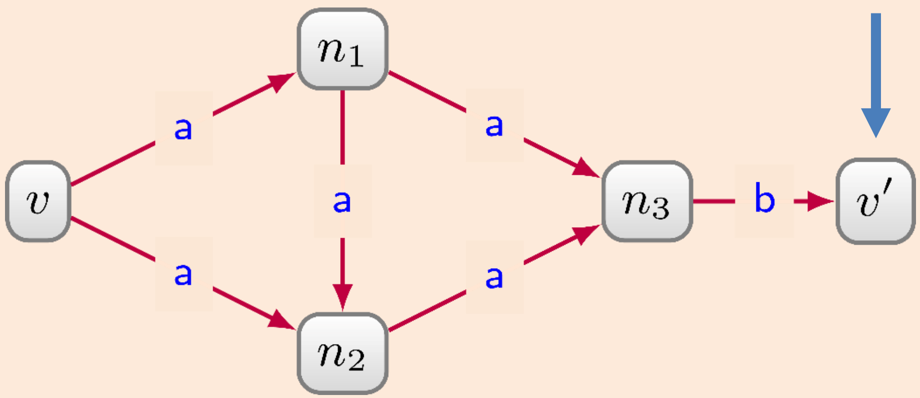


Visited:



ANY WALK $(v) =$ [a*b] $=> (?x)$

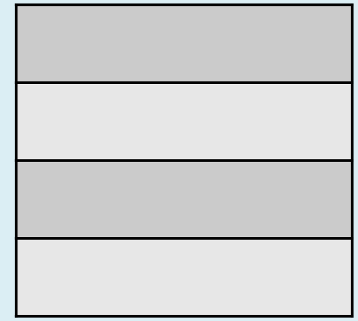




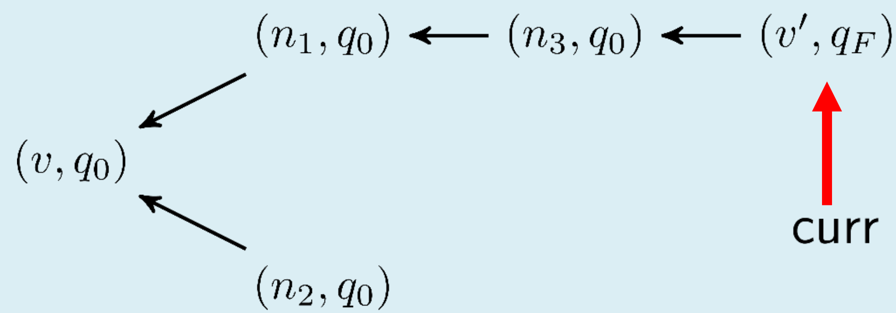
Let's see

start $\leftarrow (v, q_0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  curr=Open.pop()
  if $q == q_F$ **then**
    getPath(curr)
  for next $= (n', q') \in$ Neighbours(curr) **do**
    if !(next $\in$ Visited) **then**
      next $= (n', q', \text{curr})$
      Open.push(next)
      Visited.push(next)

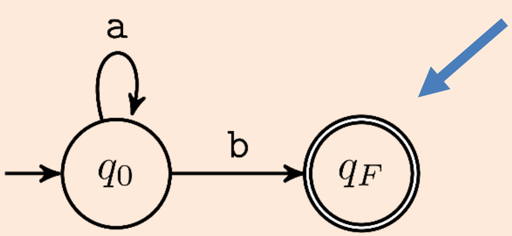
Open:

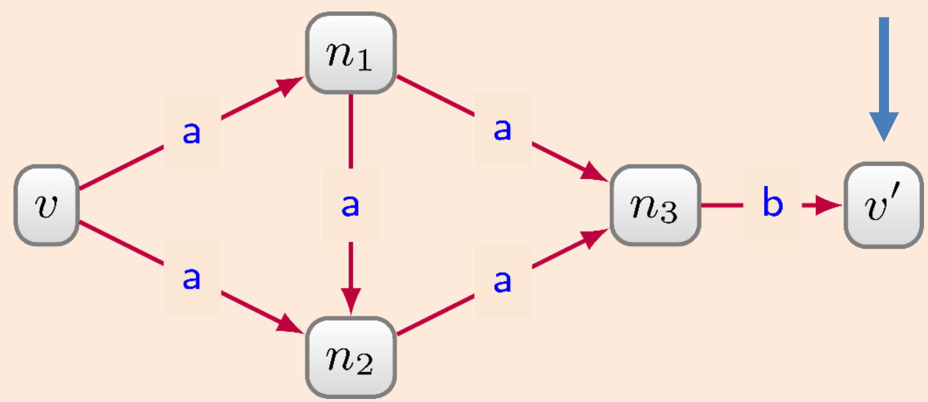


Visited:



ANY WALK $(v) = [\text{a}^*\text{b}] => (?x)$





ANY WALK

$$\text{ANY WALK } (v) = [\texttt{regex}] \texttt{=>} (?x)$$

BFS

$$\text{ANY SHORTEST WALK } (v) = [\texttt{regex}] \texttt{=>} (?x)$$

Theorem. *Let G be a graph database and q the query:*

$$\text{ANY (SHORTEST)? WALK } (v) = [\texttt{regex}] \texttt{=>} (?x)$$

Computing the output of q over G can be done with $O(|\texttt{regex}| \times |G|)$ pre-processing and output-linear delay.

Does this work in practice?

- MillenniumDB implements it:
  - Algorithm works off the bat with B+trees
  - Basically EDGE(src, type, tgt, edgeId) relation
  - Classical iterator interface
    - Results returned as soon as available
    - Algorithm pauses when a result is found

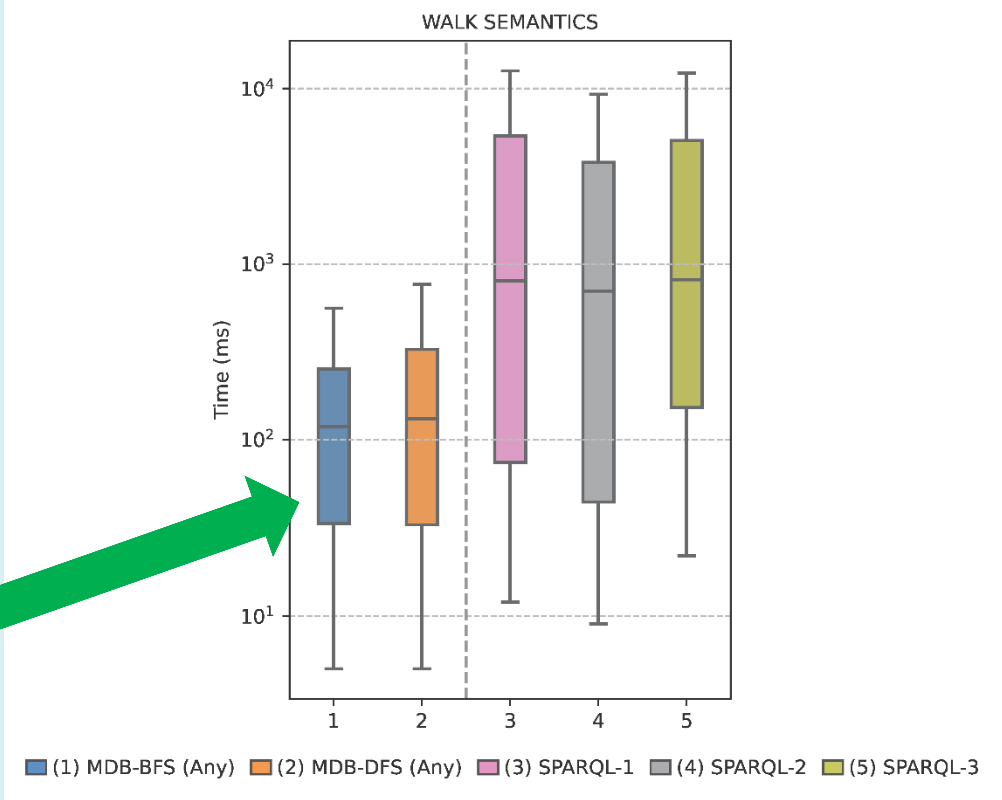
- Try it for yourself:

https://mdb.imfd.cl/path_finder/

See [MDB, FMRV23]

Does this work in practice?

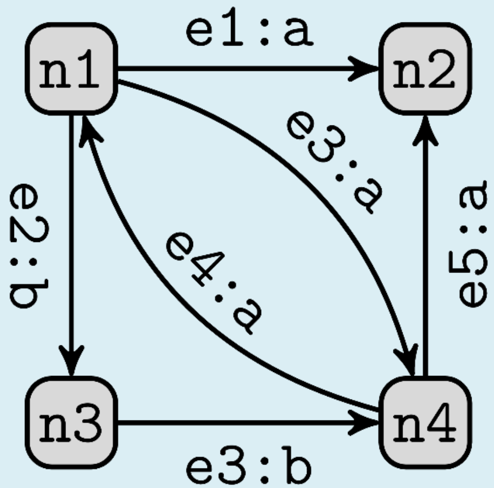
- Wikidata-based benchmark [WDBench]:
  - 1.25B edges (60000 edge labels)/300M nodes
  - 659 (non-bot) user defined queries ([MKGGB18])
  - (100,000 limit – some queries have >10M results, 1min timeout)



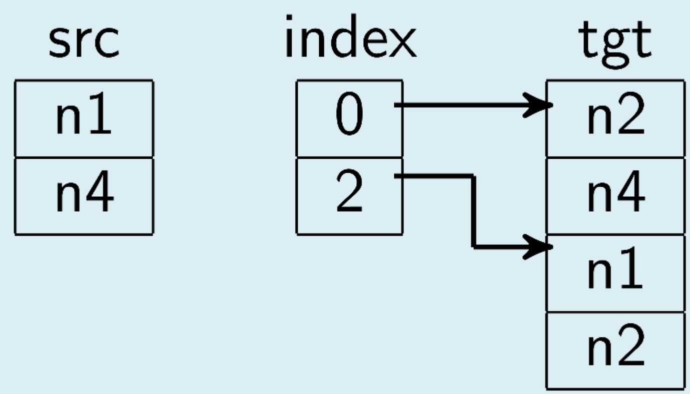
See [MDB, FMRV23]

Additional considerations 1

- CSR-based storage gives better performance [FMRV23]
  - CSRs can also be built on-the-fly as needed by the query



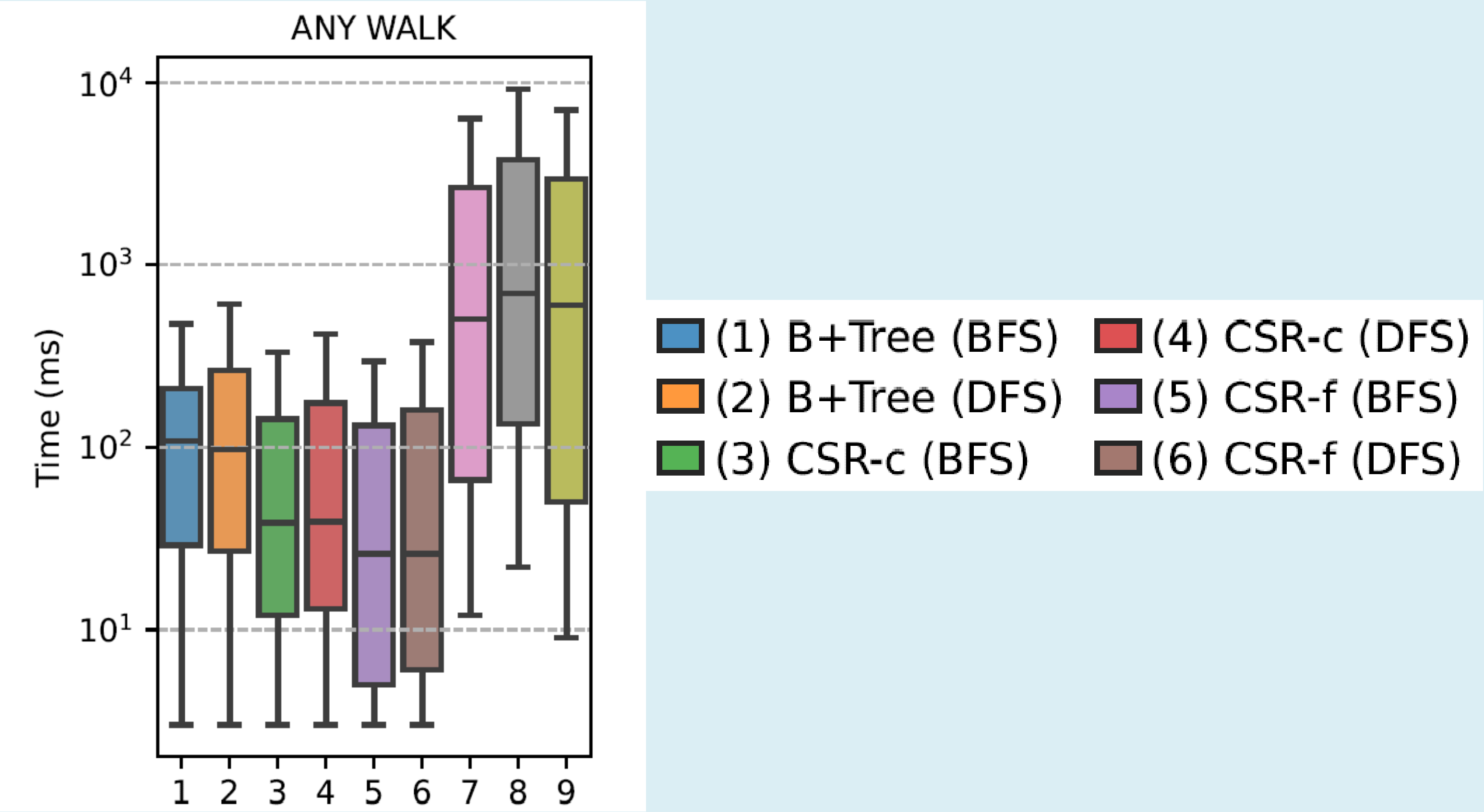
Graph database G



CSR for the label a

Additional considerations 1

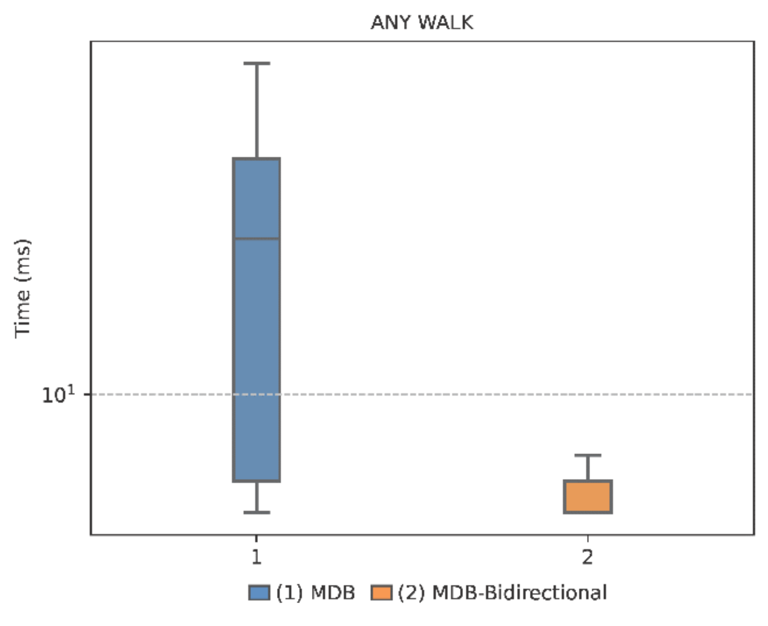
- CSR-based storage gives better performance [FMRV23]
  - CSRs can also be built on-the-fly as needed by the query



Additional considerations 2

- Significant speedups possible when both source and target are known [XVG19]
  - Basically meet-in-the-middle approach to BFS
  - This works for queries where start and end are fixed

$$\text{ANY WALK } (\texttt{start}) = [\texttt{regex}] => (\texttt{end})$$



Additional considerations 3

- We construct a compressed representation of the resulting paths [MNPRVV22]
  - Also called path multiset representation (PMR)

$$
\begin{tikzcd}
(n_0, q_0) & (n_1, q_0) \arrow[l] & (n_2, q_0) \arrow[l] & (n_3, q_0) \arrow[l] & (n_4, q_0) \arrow[l] & (n_5, q_F) \arrow[l] \\
 & (n'_1, q_F) \arrow[ul] & (n'_2, q_F) \arrow[ul] & (n'_3, q_F) \arrow[ul] & (n'_4, q_F) \arrow[ul] & (n'_5, q_F) \arrow[ul]
\end{tikzcd}
$$

$$
\begin{aligned}
&n_0 \to n'_1 \\
&n_0 \to n_1 \to n'_2 \\
&n_0 \to n_1 \to n_2 \to n'_2 \\
&n_0 \to n_1 \to n_2 \to n_3 \to n'_4 \\
&n_0 \to n_1 \to n_2 \to n_3 \to n_4 \to n'_5 \\
&n_0 \to n_1 \to n_2 \to n_3 \to n_4 \to n_5
\end{aligned}
$$

All shortest walks

ALL SHORTEST WALKS

$$\text{ALL SHORTEST WALK } (v) = [\texttt{regex}] => (?x)$$

Theorem. *Let G be a graph database and q the query:*

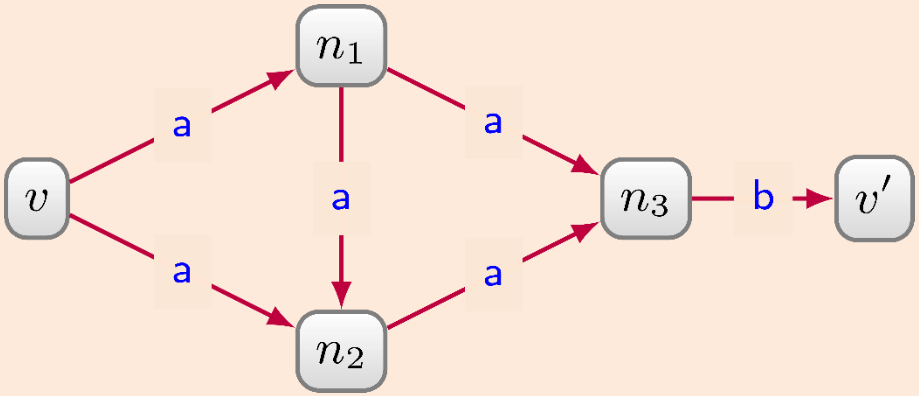
$$\text{ALL SHORTEST WALK } (v) = [\texttt{regex}] => (?x)$$

Computing the output of q over G can be done with $O(|\texttt{regex}| \times |G|)$ pre-processing and output-linear delay.

Same as ANY???

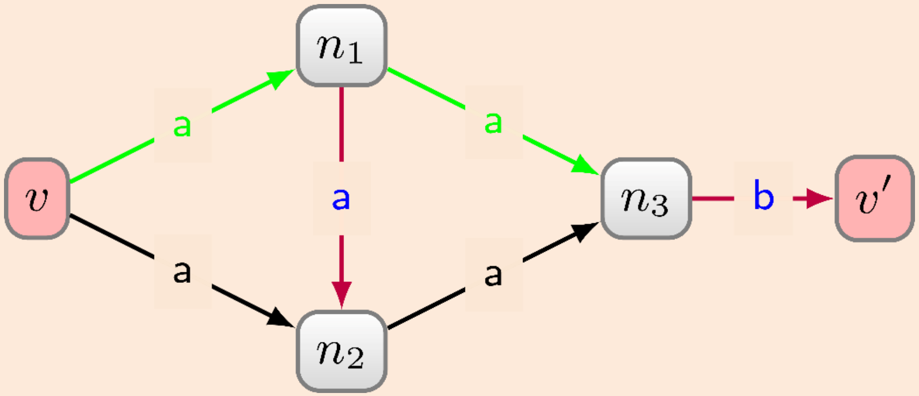
What are we looking for?

ALL SHORTEST WALK $(v) = [\mathtt{a}^*\mathtt{b}] => (?x)$



What are we looking for?

ALL SHORTEST WALK $(v) = [\mathtt{a}^*\mathtt{b}]$ => $(?x)$



Path #1: v → n1 → n3 → v’

Path #2: v → n2 → n3 → v’

How do we do this?

Similar as before:

- Graph is an automaton
- Regular expression is an automaton
- Build the product graph
- Start searching for **all shortest paths**
  - From the start node
  - Till hitting a node tagged by an end state of the automaton

How do we find all shortest paths between two nodes?

All shortest paths

Let us do this for normal graphs:

- G = (V,E)
- Fix a node v
- For v' reachable from v: enumerate **all shortest paths**

We use BFS:

- But we will allow revisiting nodes
  - When this is done by another shortest path
  - We will need to record the shortest path length
  - And allow a revisit when the length is the same

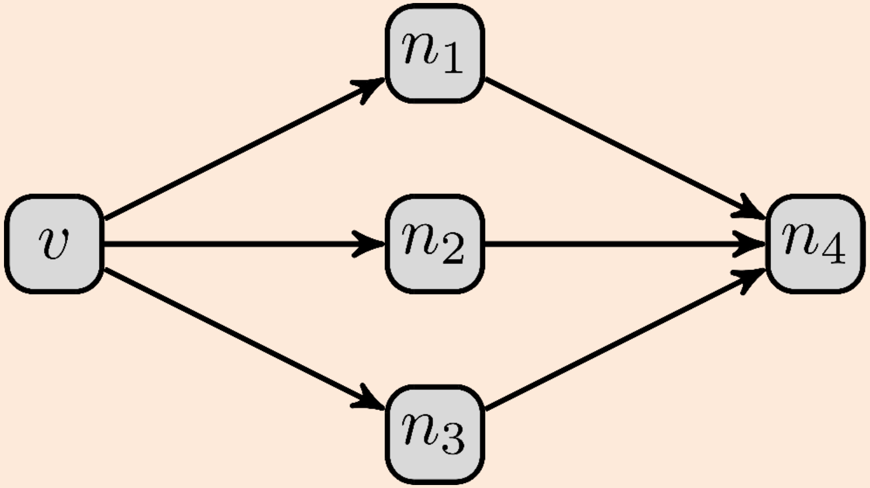
BFS – all shortest paths

Algorithm 4 All shortest paths reachable from v in $G = (V, E)$

```
function AllShortest(G, v)
    Open.init()                                          ▷ Empty queue
    Visited.init()                                       ▷ Empty dictionary
    start ← (v, 0, ⊥)
    Open.push(start)
    Visited.push(start)
    while !Open.isEmpty() do
        current=Open.pop()                               ▷ current = (n, depth, prevList)
        enumeratePaths(current)                          ▷ Enumerate all shortest paths
        for n′ s.t. (n, n′) ∈ E do
            if !(n′ ∈ Visited) then
                new = (n′, depth + 1, prevList.init(current))
                Open.push(new)
                Visited.push(new)
            if n′ ∈ Visited then
                new = Visited.get(n′)                    ▷ new = (n′, depth′, prevList’)
                if depth′ == depth + 1 then              ▷ Another shortest path to n′
                    prevList’.add(current)
```

Let's see

Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList$)$
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            $(n', d',$ prevList'$)$ = Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)

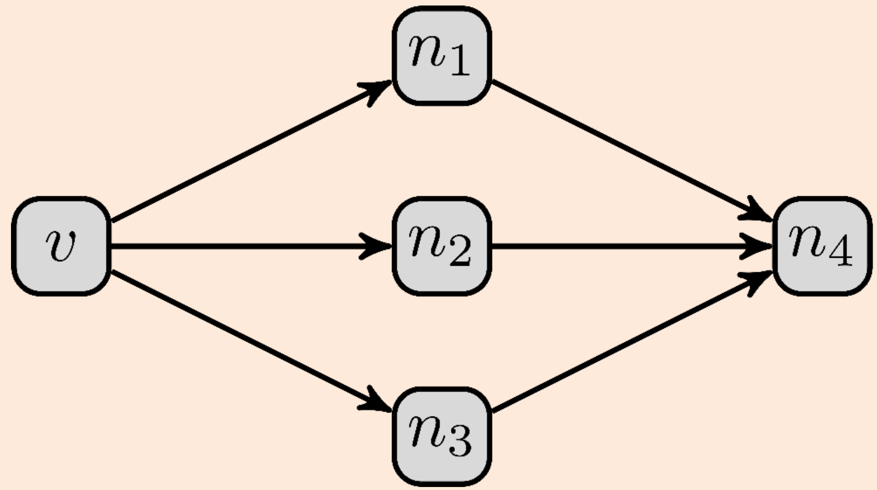


Open:

Visited:

Let's see

Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  current=Open.pop()
  enumeratePaths(current)
  for n' s.t. $(n, n') \in E$ **do**
    if !($n' \in$ Visited) **then**
      prevList.init(current)
      new $= (n', depth + 1,$ prevList$)$
      Open.push(new)
      Visited.push(new)
    if $n' \in$ Visited **then**
      $(n', d',$ prevList'$) =$ Visited.get(n')
      if $d' == depth + 1$ **then**
        prevList'.add(current)



Open:

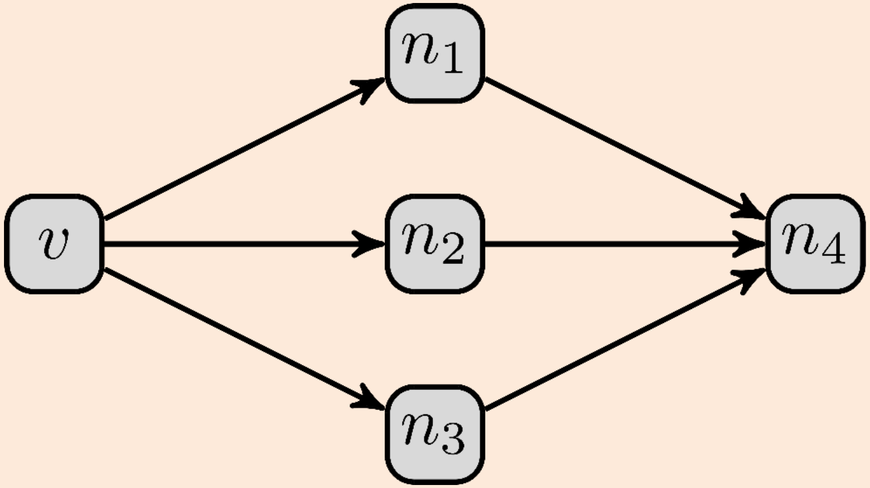
$v, 0,$ pL$_v$		

Visited:

$(v, 0)$

Let's see

Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList)
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            (n', d', prevList') = Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



Open:

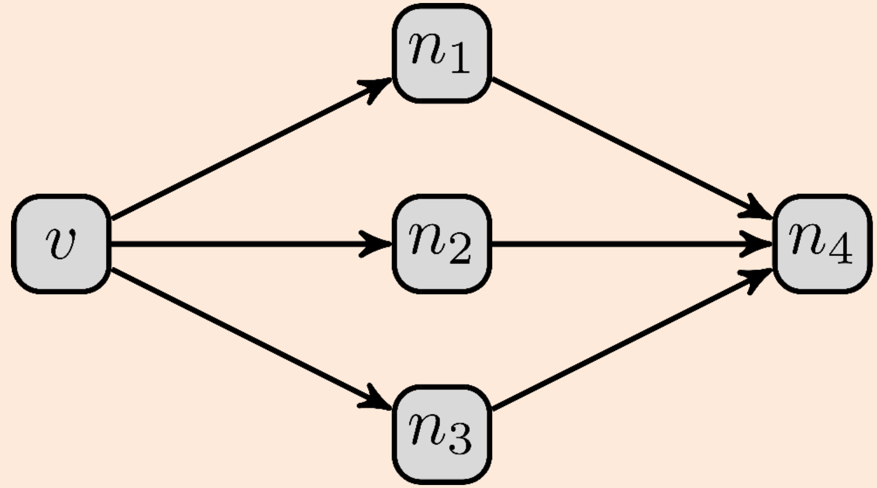
$v, 0,$ pL$_v$		

Visited:

$(v, 0)$

Let's see

Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  current=Open.pop()
  enumeratePaths(current)
  for n' s.t. $(n, n') \in E$ **do**
    if !($n' \in$ Visited) **then**
      prevList.init(current)
      new $= (n', depth + 1,$ prevList)
      Open.push(new)
      Visited.push(new)
    if $n' \in$ Visited **then**
      $(n', d',$ prevList') = Visited.get(n')
      if $d' == depth + 1$ **then**
        prevList'.add(current)



Open:

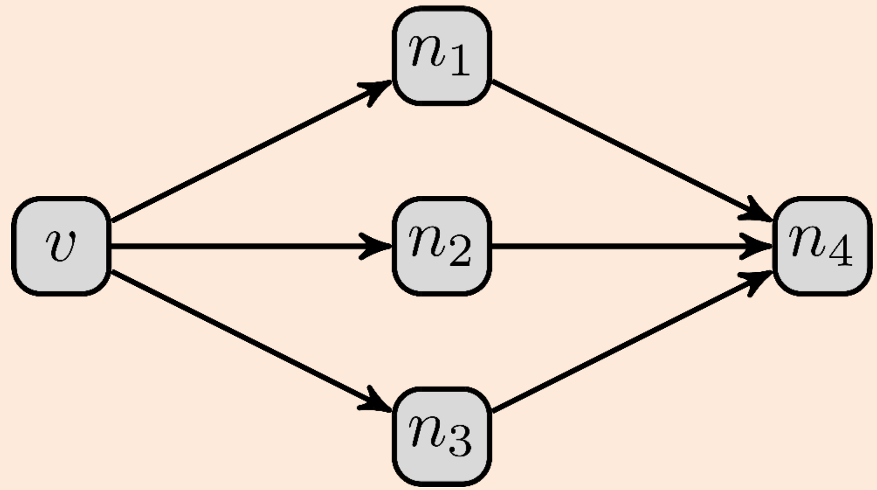
Visited:

$(v, 0)$

current

Let's see

Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if $!(n' \in$ Visited$)$ **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList$)$
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            $(n', d',$ prevList'$) =$ Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



Open:

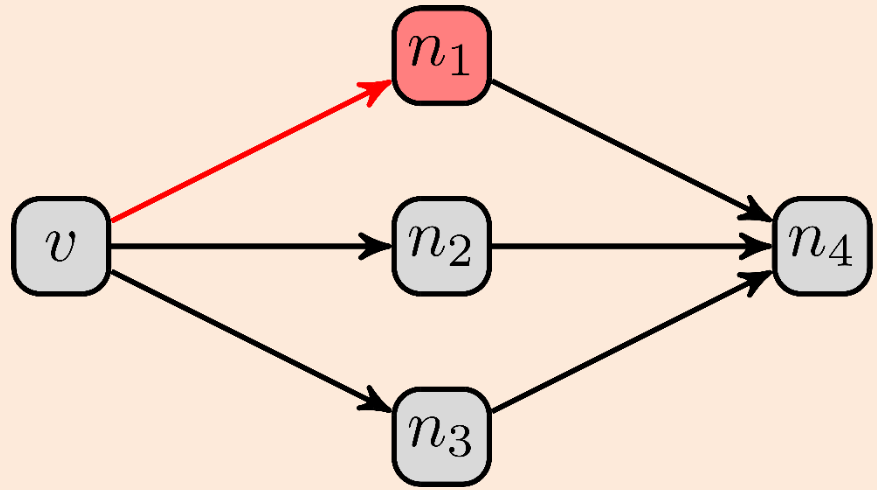
Visited:

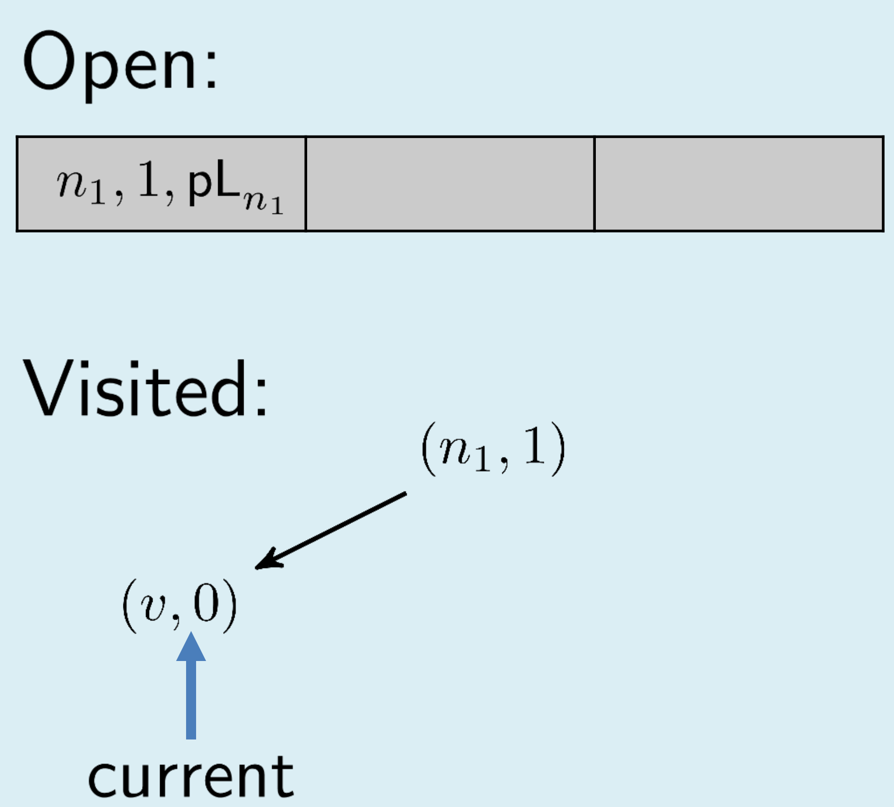
$(v, 0)$

current

Let's see

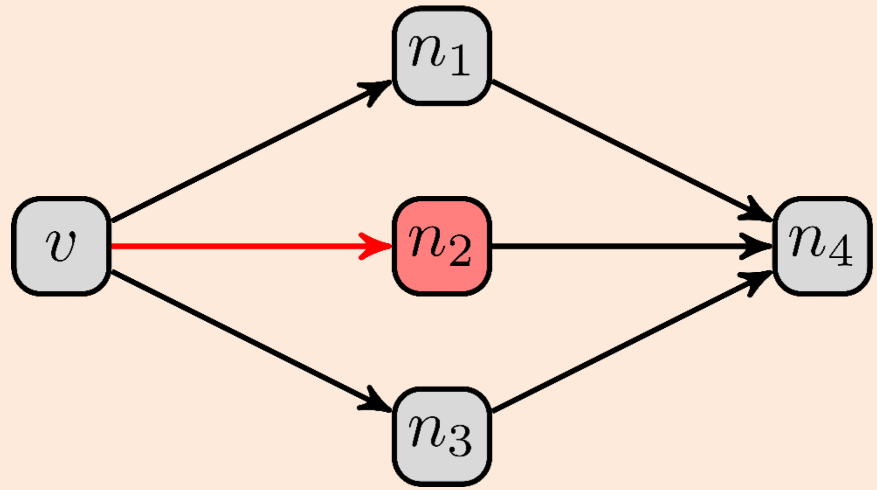
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList$)$
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            $(n', d',$ prevList'$) =$ Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)





Let's see

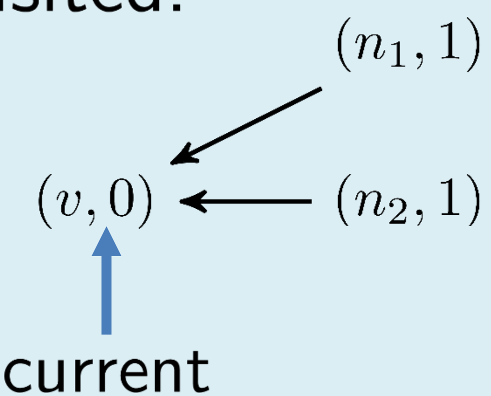
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList)
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            $(n', d',$ prevList') $=$ Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



Open:

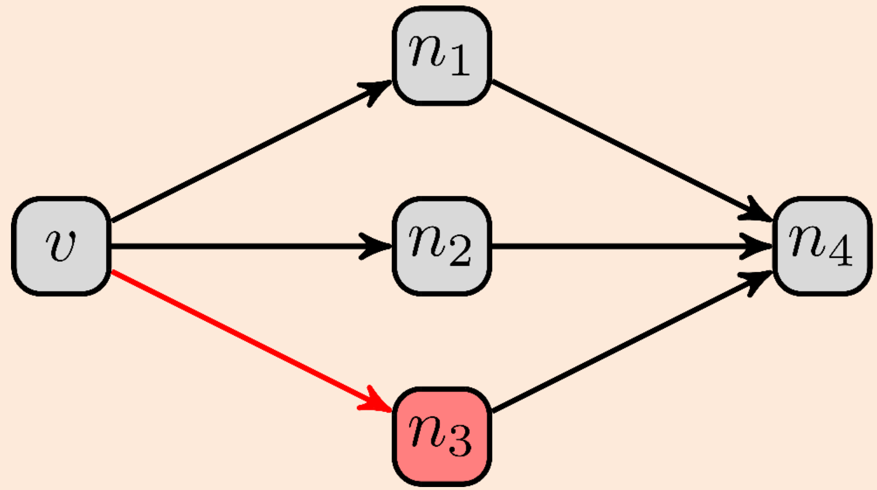
$n_1, 1,$ pL$_{n_1}$	$n_2, 1,$ pL$_{n_2}$	

Visited:



Let's see

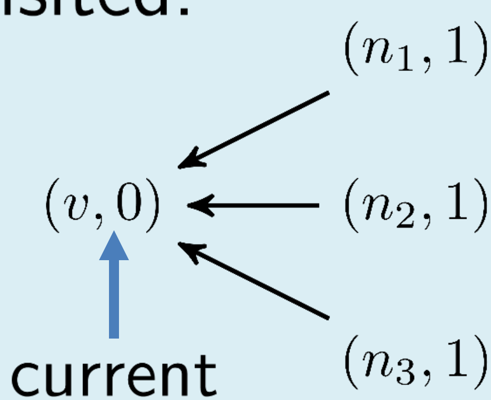
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList)
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            $(n', d',$ prevList') = Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



Open:

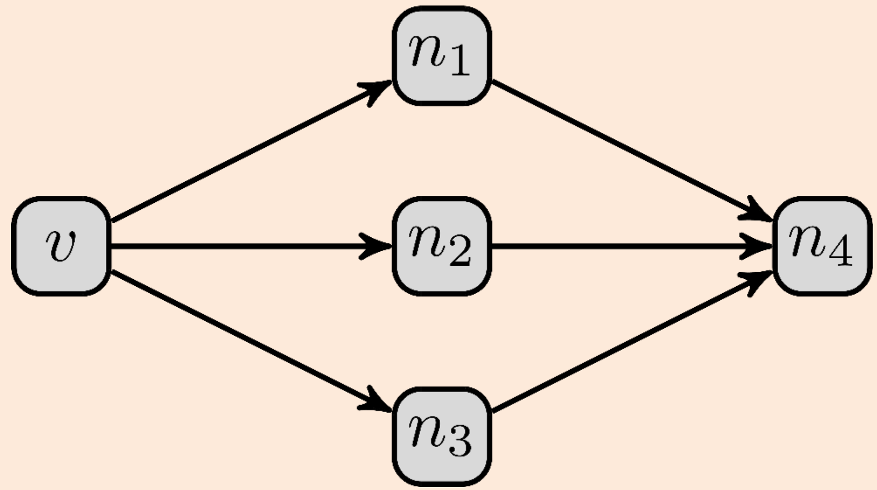
$n_1, 1, \mathrm{pL}_{n_1}$	$n_2, 1, \mathrm{pL}_{n_2}$	$n_3, 1, \mathrm{pL}_{n_3}$

Visited:



Let's see

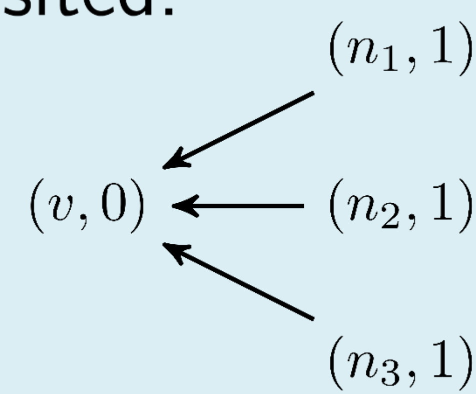
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList$)$
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            $(n', d',$ prevList'$) =$ Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



Open:

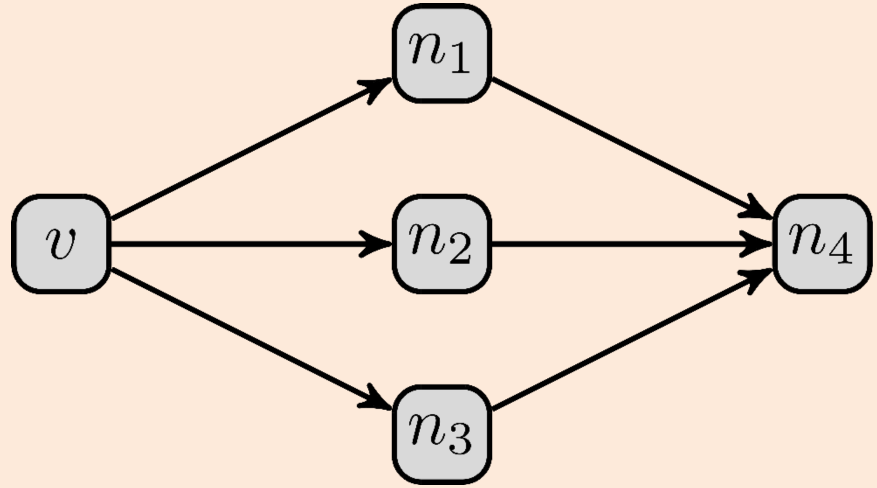
$n_1, 1,$ pL$_{n_1}$	$n_2, 1,$ pL$_{n_2}$	$n_3, 1,$ pL$_{n_3}$

Visited:



Let's see

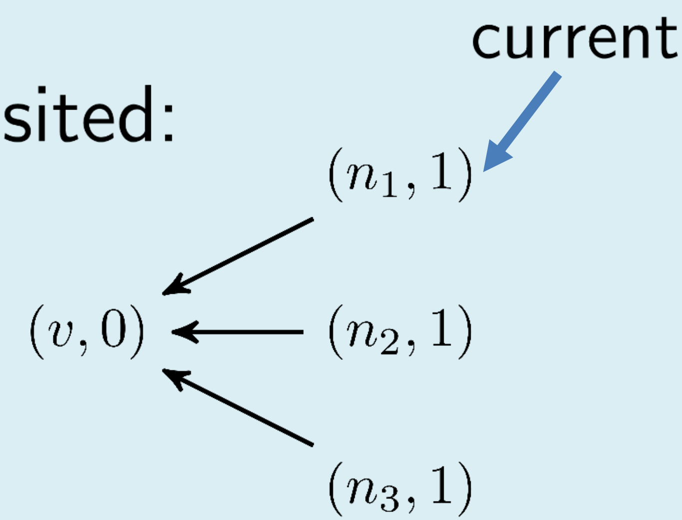
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList$)$
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            $(n', d',$ prevList'$) =$ Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



Open:

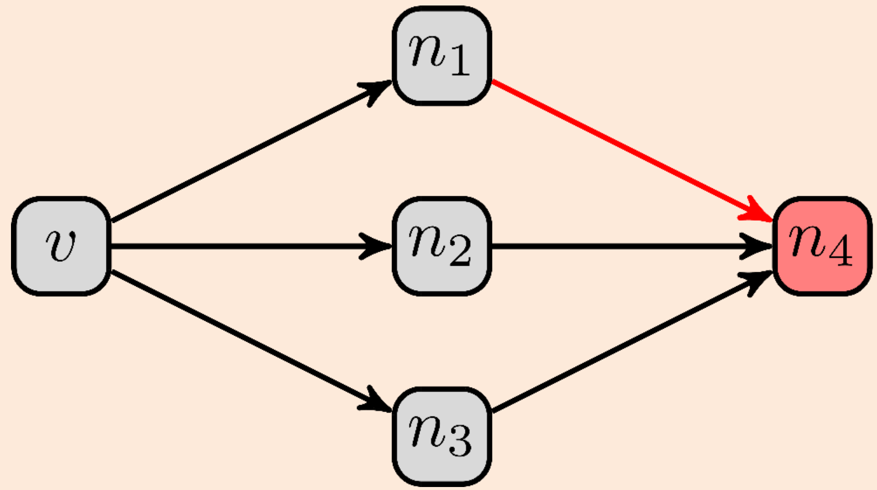
$n_2, 1, \mathsf{pL}_{n_2}$	$n_3, 1, \mathsf{pL}_{n_3}$	

Visited:



Let's see

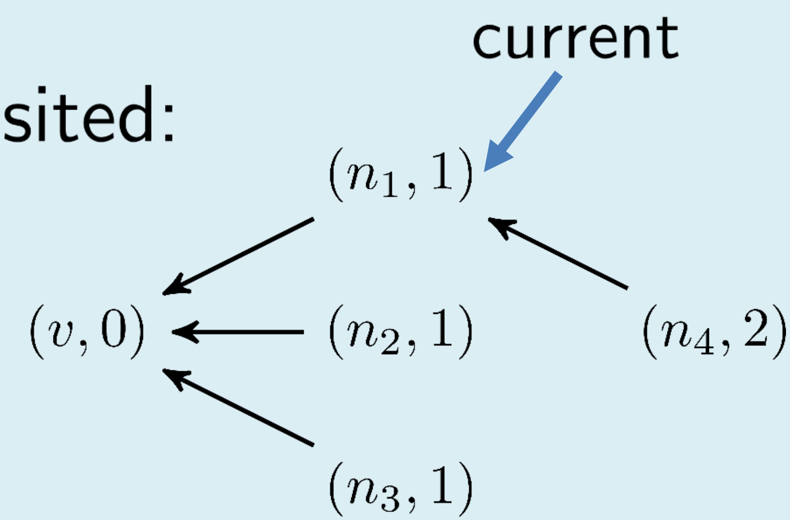
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !$(n' \in$ Visited$)$ **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList$)$
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            $(n', d',$ prevList'$) =$ Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



Open:

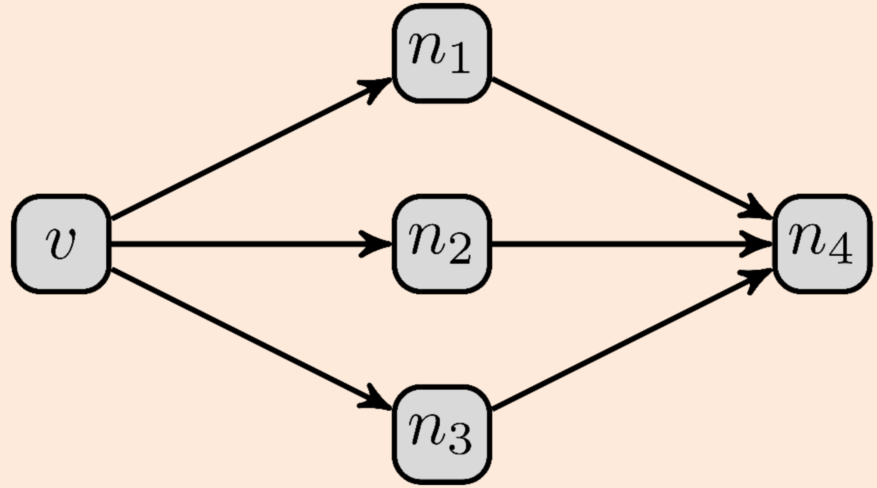
$n_2, 1, \mathrm{pL}_{n_2}$	$n_3, 1, \mathrm{pL}_{n_3}$	$n_4, 3, \mathrm{pL}_{n_4}$

Visited:



Let's see

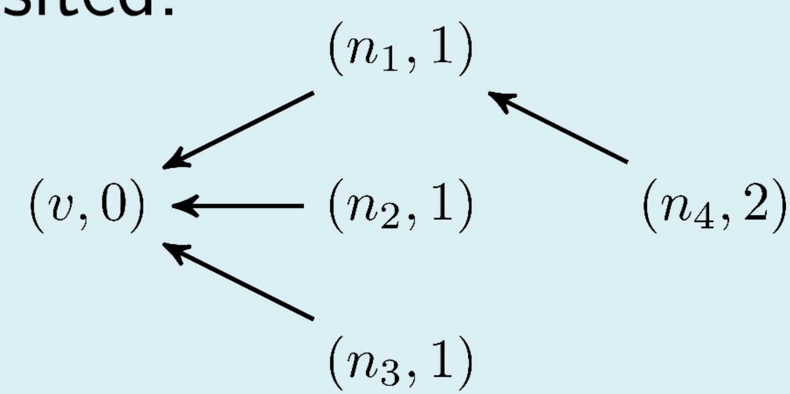
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  current=Open.pop()
  enumeratePaths(current)
  for n' s.t. $(n, n') \in E$ **do**
    if !($n' \in$ Visited) **then**
      prevList.init(current)
      new $= (n', depth + 1,$ prevList$)$
      Open.push(new)
      Visited.push(new)
    if $n' \in$ Visited **then**
      $(n', d',$ prevList'$)$ = Visited.get(n')
      if $d' == depth + 1$ **then**
        prevList'.add(current)



Open:

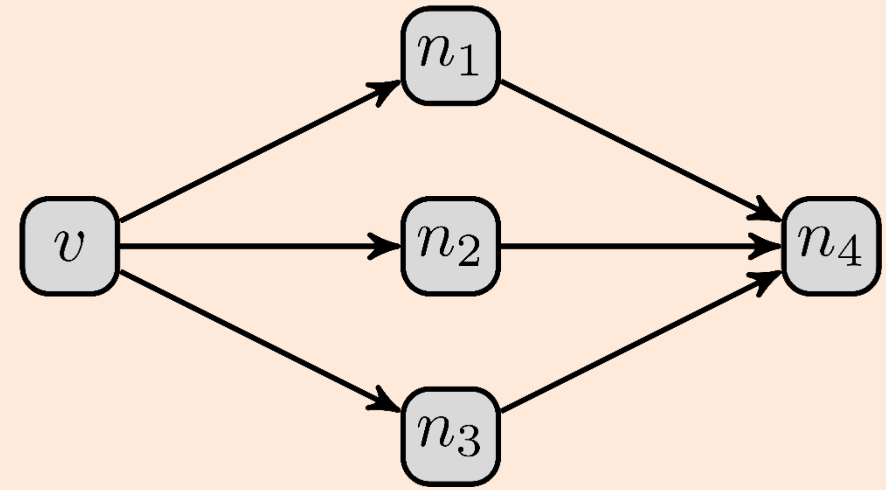
$n_2, 1, \text{pL}_{n_2}$	$n_3, 1, \text{pL}_{n_3}$	$n_4, 3, \text{pL}_{n_4}$

Visited:



Let's see

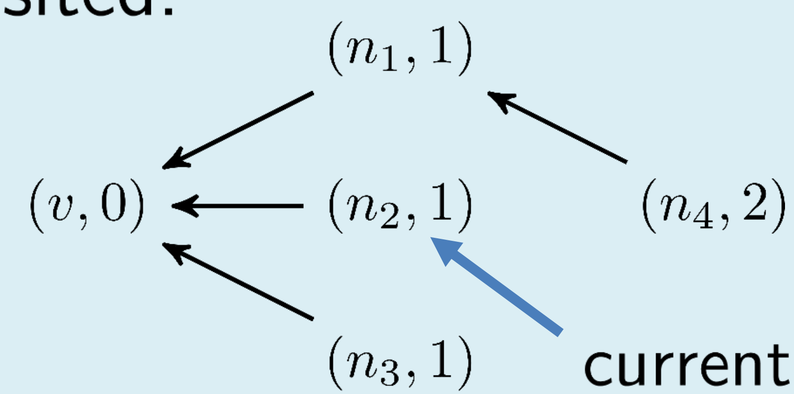
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  current=Open.pop()
  enumeratePaths(current)
  for n' s.t. $(n, n') \in E$ **do**
    if !($n' \in$ Visited) **then**
      prevList.init(current)
      new $= (n', depth + 1,$ prevList$)$
      Open.push(new)
      Visited.push(new)
    if $n' \in$ Visited **then**
      $(n', d',$ prevList'$) =$ Visited.get(n')
      if $d' == depth + 1$ **then**
        prevList'.add(current)



Open:

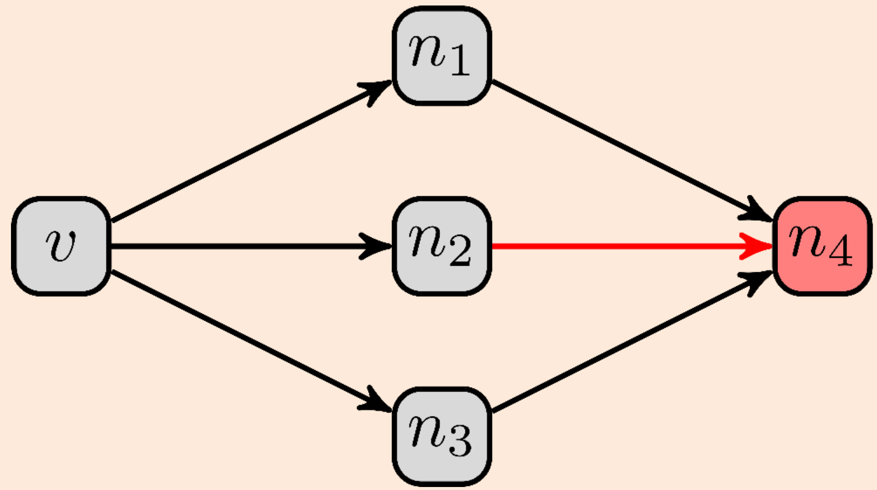
$n_3, 1, \text{pL}_{n_3}$	$n_4, 3, \text{pL}_{n_4}$	

Visited:



Let's see

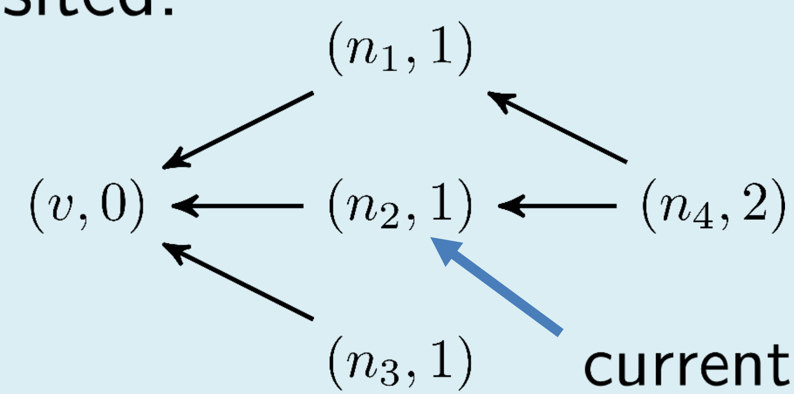
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList)
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            (n', d', prevList') = Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



Open:

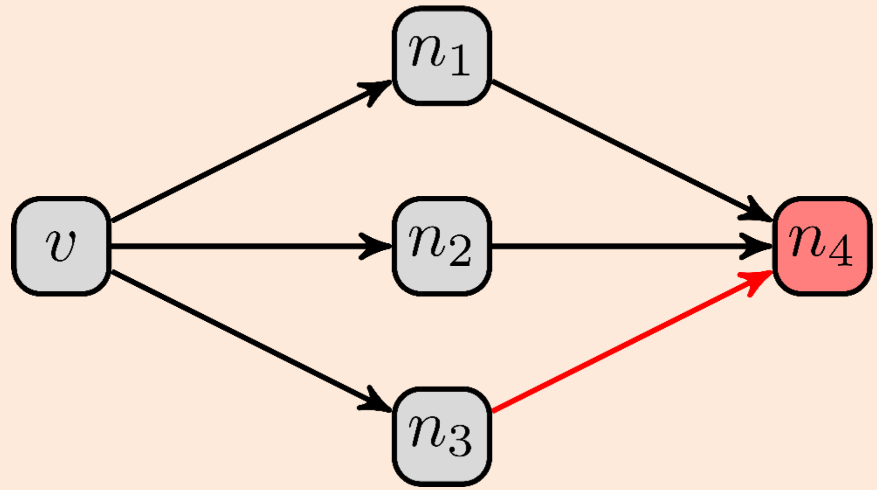
$n_3, 1, \mathrm{pL}_{n_3}$	$n_4, 3, \mathrm{pL}_{n_4}$	

Visited:



Let's see

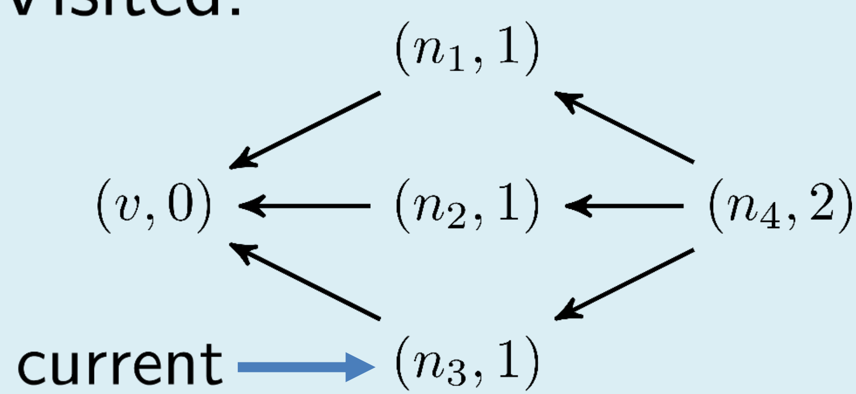
Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
  current=Open.pop()
  enumeratePaths(current)
  for n' s.t. $(n, n') \in E$ **do**
    if !($n' \in$ Visited) **then**
      prevList.init(current)
      new $= (n', depth + 1,$ prevList)
      Open.push(new)
      Visited.push(new)
    if $n' \in$ Visited **then**
      (n', d', prevList') = Visited.get(n')
      if $d' == depth + 1$ **then**
        prevList'.add(current)



Open:

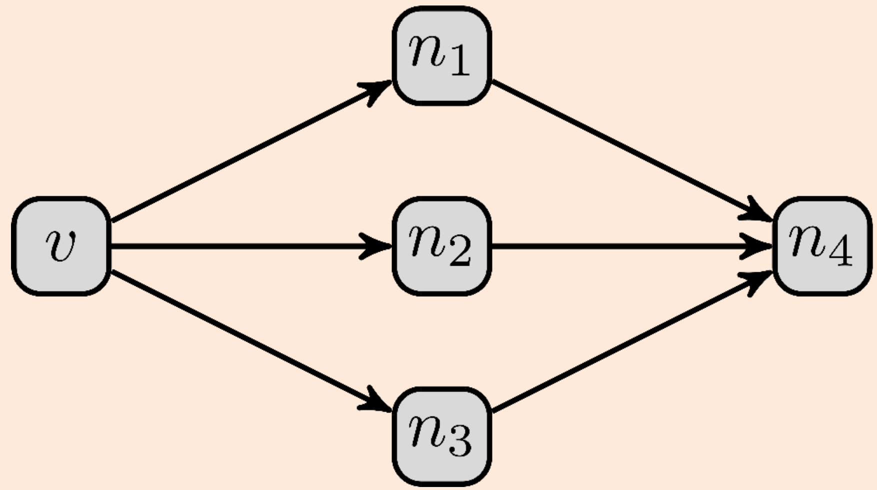
$n_4, 3,$ pL$_{n_4}$		

Visited:



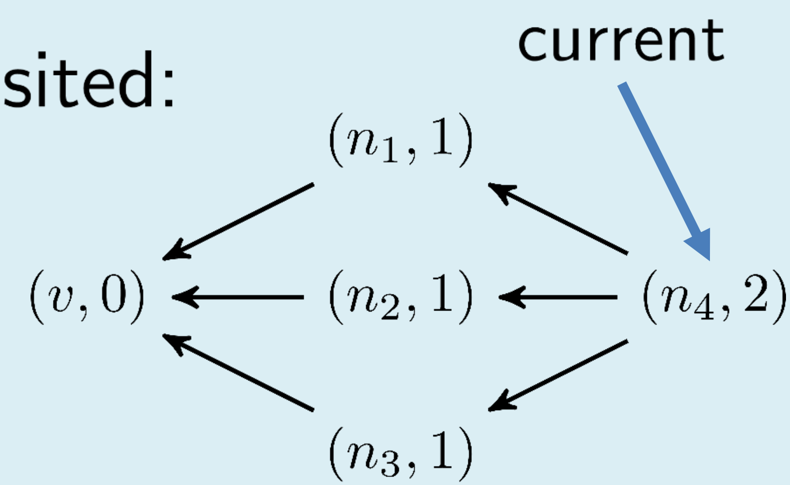
Let's see

Open.init()
Visited.init()
start $\leftarrow (v, 0, \perp)$
Open.push(start)
Visited.push(start)
while !Open.isEmpty() **do**
    current=Open.pop()
    enumeratePaths(current)
    for n' s.t. $(n, n') \in E$ **do**
        if !($n' \in$ Visited) **then**
            prevList.init(current)
            new $= (n', depth + 1,$ prevList)
            Open.push(new)
            Visited.push(new)
        if $n' \in$ Visited **then**
            (n', d', prevList') = Visited.get(n')
            if $d' == depth + 1$ **then**
                prevList'.add(current)



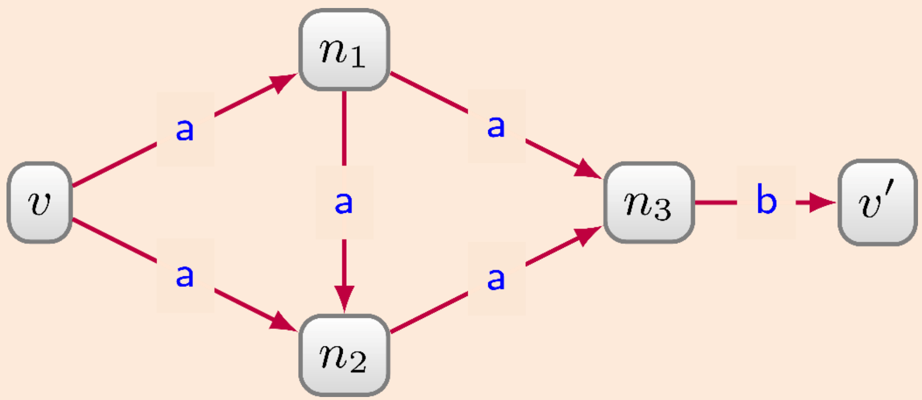
Open:

Visited:



What about these guys?

ALL SHORTEST WALK (v)-[a^*b]->(?x)



Same as before [V22]:

- Run the algorithm on the product graph
- From the start node (v,q_0)
- Needs some assumptions (automaton unambiguous)

Basically

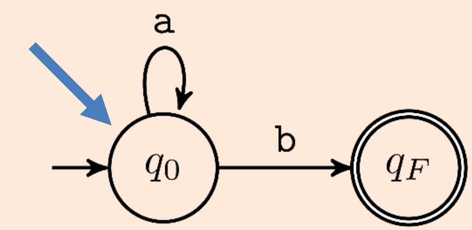
Algorithm 1 Evaluation algorithm for a graph database G and an RPQ $query$ = ALL SHORTEST WALK $(v, \texttt{regex}, ?x)$.

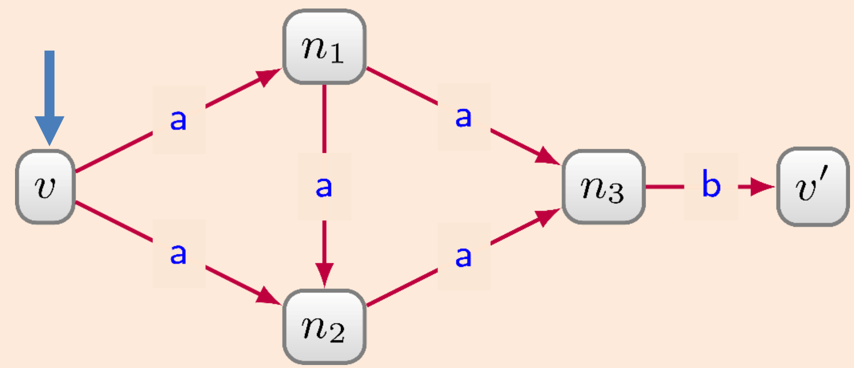
```
function Search(G, query)
    A ← Automaton(regex)
    Open.init()                                        ▷ Queue
    Visited.init()                                     ▷ Dictionary on (n, q)
    startState ← (v, q_0, 0, ⊥)
    Visited.push(startState)
    Open.push(startState)
    while !Open.isEmpty() do
        current ← Open.pop()                           ▷ current = (n, q, depth,prevList)
        if q == q_F then
            enumAllShortestPaths(current)
        for next = (n', q') ∈ Neighbors(current, G, A) do
            if (n', q', *, *) ∈ Visited then
                (n', q', depth',prevList') ← Visited.get(n', q')
                if depth + 1 == depth' then
                    prevList'.add(current)
            else
                prevList.init()
                prevList.add(current)
                newState ← (n', q', depth + 1,prevList)
                Visited.push(newState)
                Open.push(newState)
```

Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    if $q == q_F$ **then**
        enumAllShortestPaths(current)
    for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
        if $(n', q', *, *) \in$ Visited **then**
            $(n', q', depth'$,prevList'$) \leftarrow$ Visited.get(n', q')
            if $depth + 1 == depth'$ **then**
                prevList'.add(current)
        else
            prevList.init()
            prevList.add(current)
            newState $\leftarrow (n', q', depth + 1,$prevList$)$
            Visited.push(newState)
            Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$





Open:

$(v, q_0, 0, \mathsf{pl}_v)$

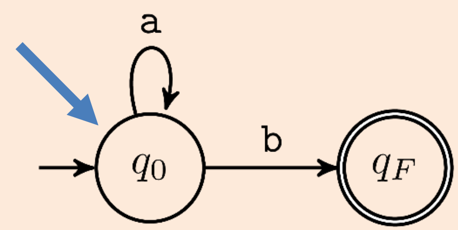
Visited:

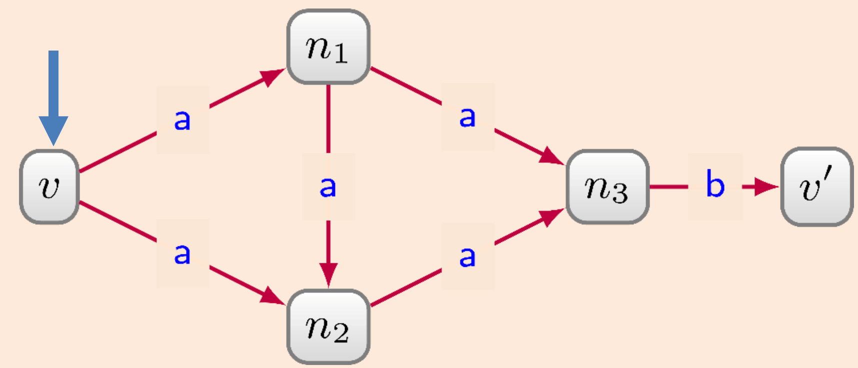
$(v, q_0, 0)$

Let's see

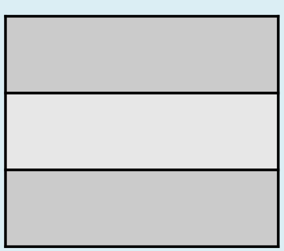
Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$)$ $\leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1,$prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$





Open:



Visited:

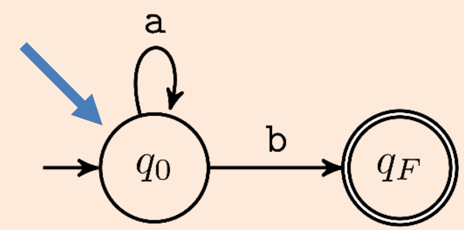
$(v, q_0, 0)$

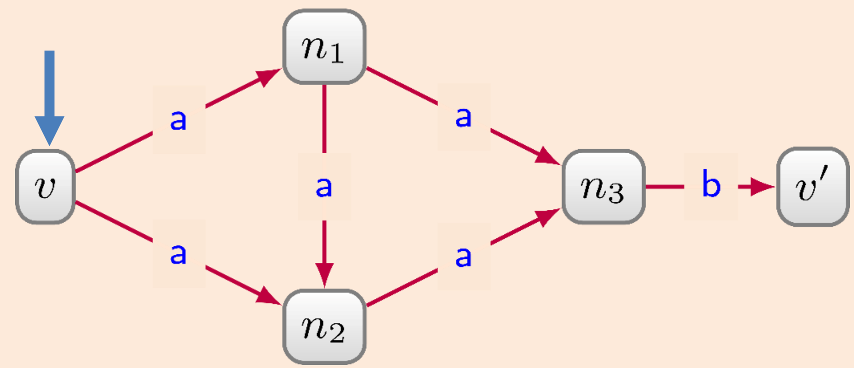
curr

Let's see

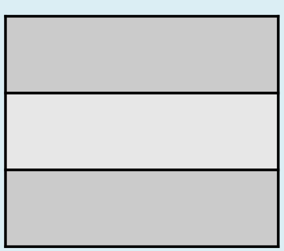
Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$)$ $\leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1$,prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$





Open:



Visited:

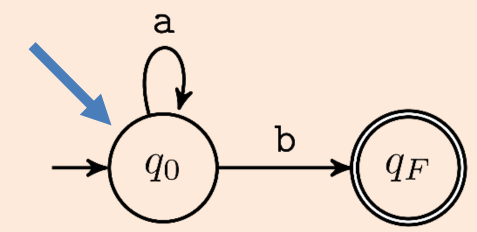
$(v, q_0, 0)$

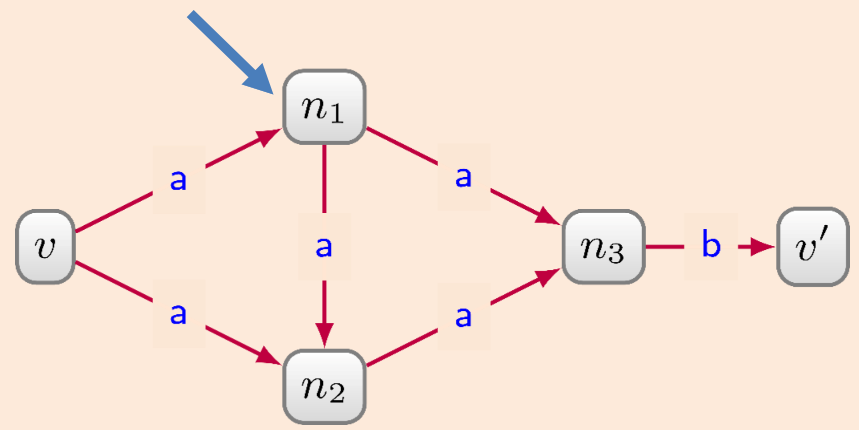
curr

Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    if $q == q_F$ **then**
        enumAllShortestPaths(current)
    for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
        if $(n', q', *, *) \in$ Visited **then**
            $(n', q', depth'$,prevList') $\leftarrow$ Visited.get(n', q')
            if $depth + 1 == depth'$ **then**
                prevList'.add(current)
        else
            prevList.init()
            prevList.add(current)
            newState $\leftarrow (n', q', depth + 1$,prevList)
            Visited.push(newState)
            Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$





Open:

Visited:

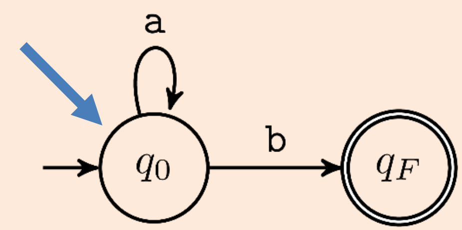
$(v, q_0, 0)$

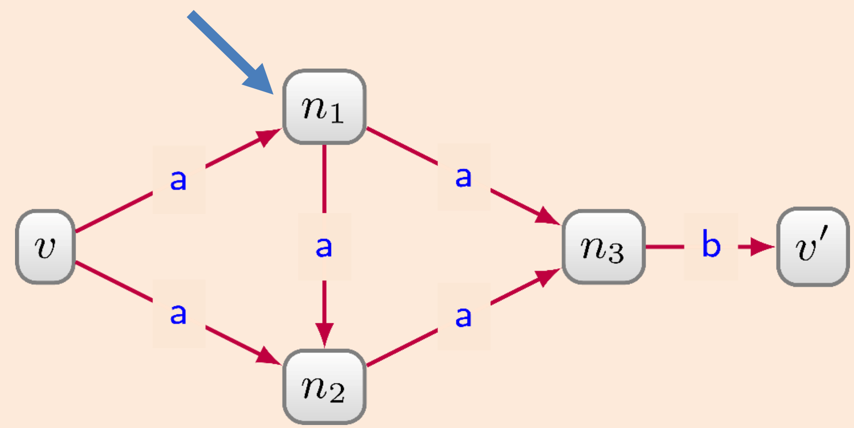
curr

Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    if $q == q_F$ **then**
        enumAllShortestPaths(current)
    for next $= (n', q') \in$ Neighbors(current, G, $\mathcal{A}$) **do**
        if $(n', q', *, *) \in$ Visited **then**
            $(n', q', depth'$,prevList'$)$ $\leftarrow$ Visited.get(n', q')
            if $depth + 1 == depth'$ **then**
                prevList'.add(current)
        else
            prevList.init()
            prevList.add(current)
            newState $\leftarrow (n', q', depth + 1,$prevList$)$
            Visited.push(newState)
            Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

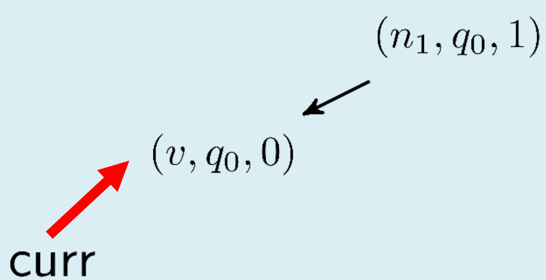




Open:

$(n_1, q_0, 1, \text{pl}_{n_1})$

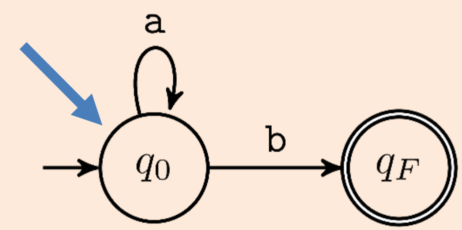
Visited:

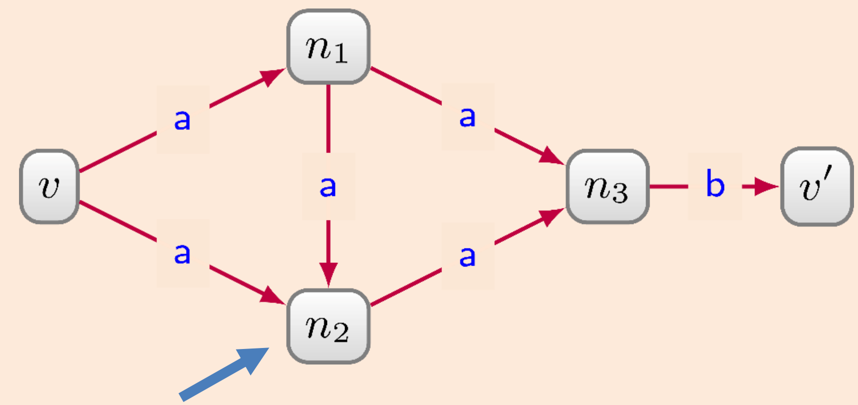


Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$)$ $\leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1$,prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

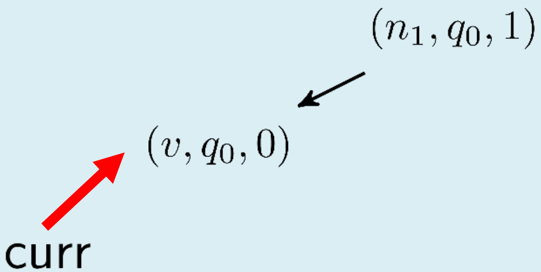




Open:

$(n_1, q_0, 1, \mathsf{pl}_{n_1})$

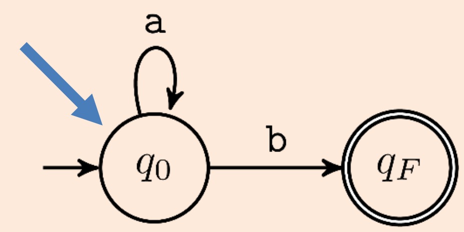
Visited:

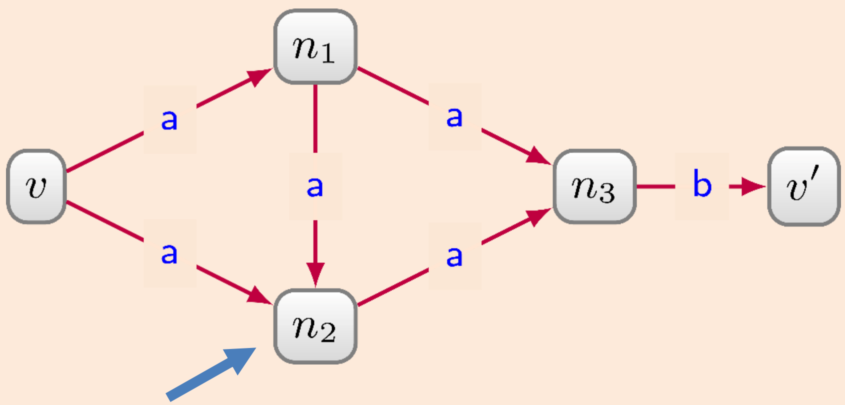


Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$) \leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1$,prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

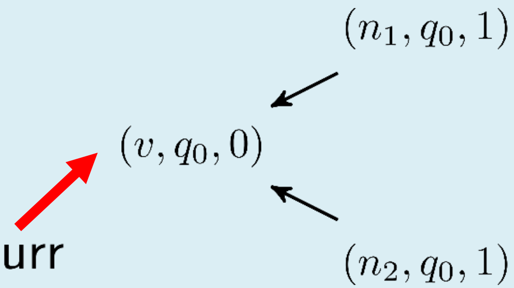




Open:

$(n_1, q_0, 1, \text{pl}_{n_1})$
$(n_2, q_0, 1, \text{pl}_{n_2})$

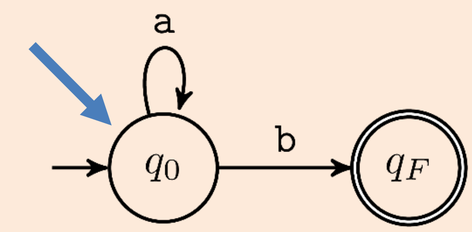
Visited:

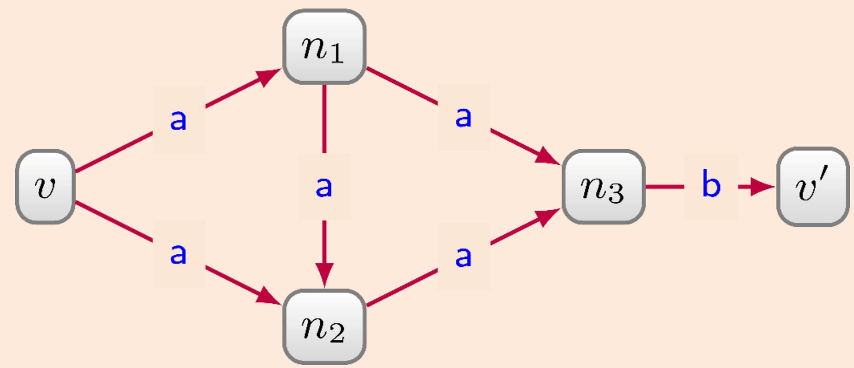


Let's see

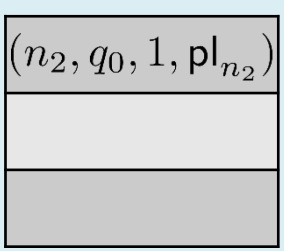
Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$)$ $\leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1,$prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

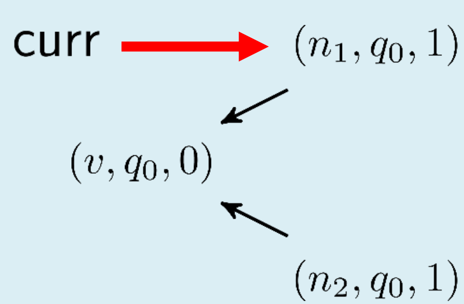




Open:



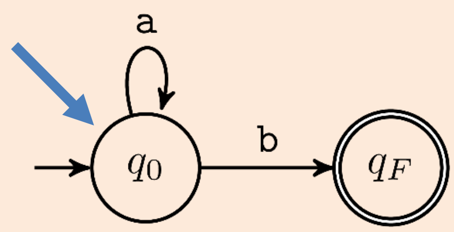
Visited:

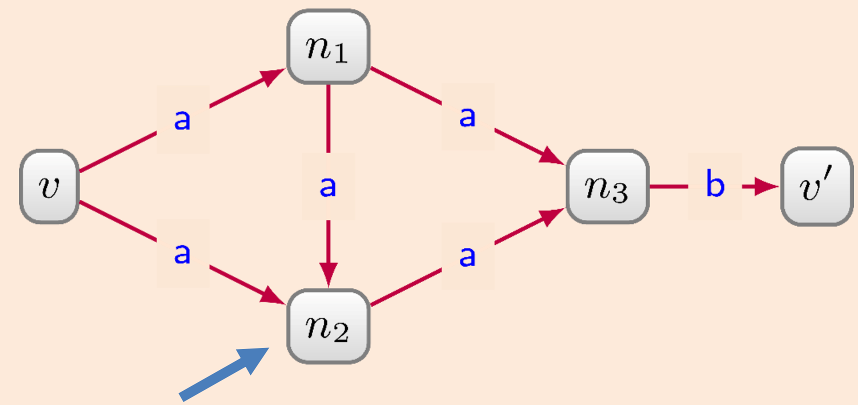


Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth',$prevList'$) \leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1,$prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$





Open:

$(n_2, q_0, 1, \text{pl}_{n_2})$

Visited:

curr $\rightarrow (n_1, q_0, 1)$

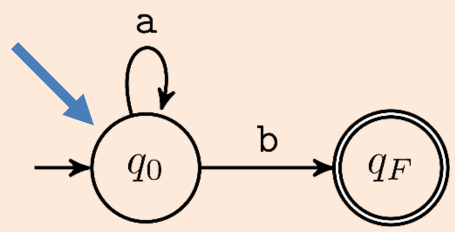
$(v, q_0, 0)$

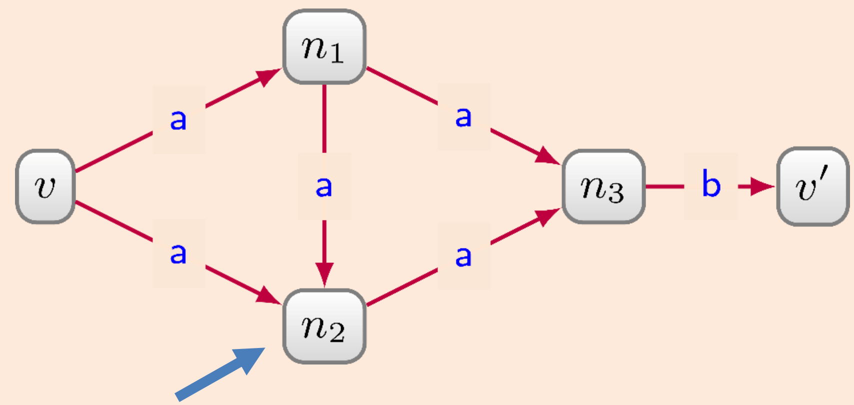
$(n_2, q_0, 1)$

Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$)$ $\leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1,$prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$





Open:

$(n_2, q_0, 1, \text{pl}_{n_2})$

Visited:

curr $\rightarrow$ $(n_1, q_0, 1)$

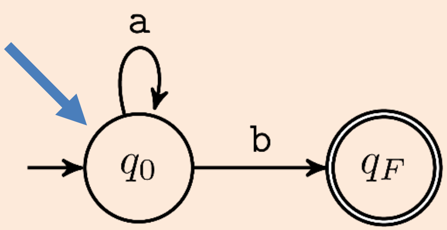
$(v, q_0, 0)$

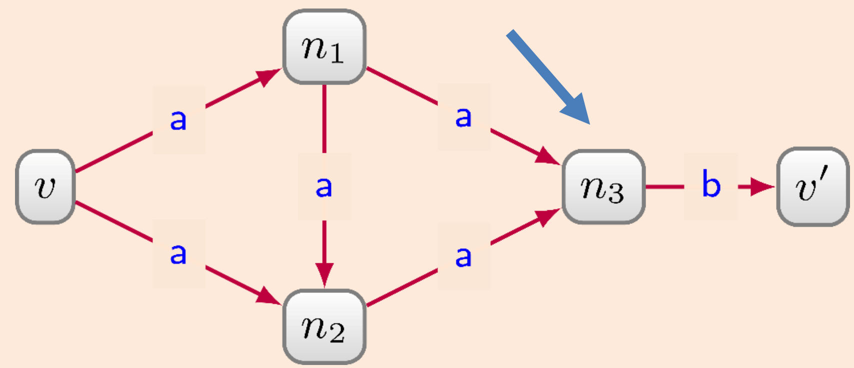
$(n_2, q_0, 1)$

Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    if $q == q_F$ **then**
        enumAllShortestPaths(current)
    for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
        if $(n', q', *, *) \in$ Visited **then**
            $(n', q', depth'$,prevList'$)$ $\leftarrow$ Visited.get(n', q')
            if $depth + 1 == depth'$ **then**
                prevList'.add(current)
        else
            prevList.init()
            prevList.add(current)
            newState $\leftarrow (n', q', depth + 1$,prevList$)$
            Visited.push(newState)
            Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

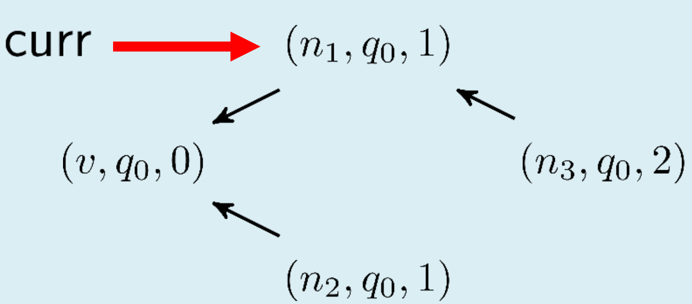




Open:

$(n_2, q_0, 1, \mathsf{pl}_{n_2})$
$(n_3, q_0, 2, \mathsf{pl}_{n_3})$

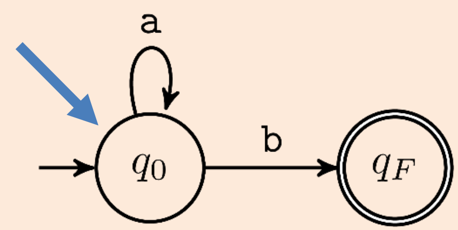
Visited:

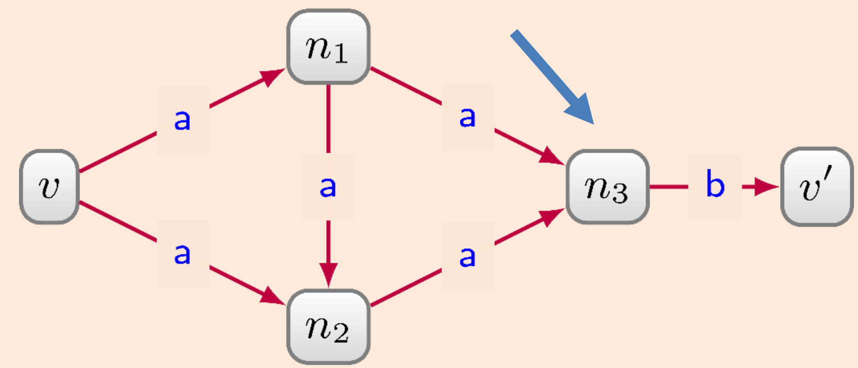


Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$) \leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1$,prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

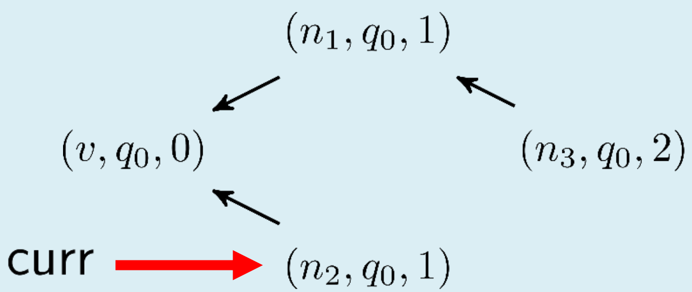




Open:

$(n_3, q_0, 2, \mathsf{pl}_{n_3})$

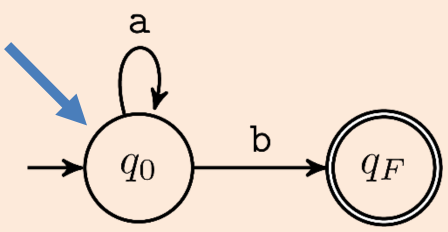
Visited:

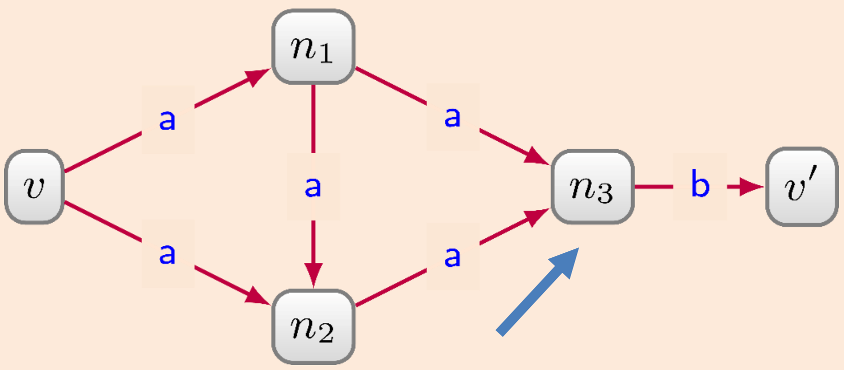


Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$) \leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1$,prevList)
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

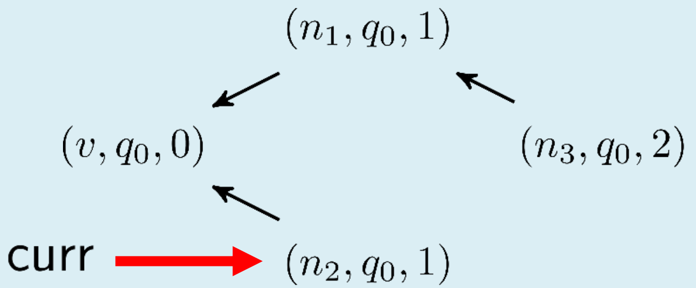




Open:

$(n_3, q_0, 2, \mathsf{pl}_{n_3})$

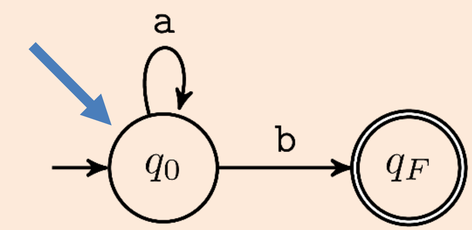
Visited:

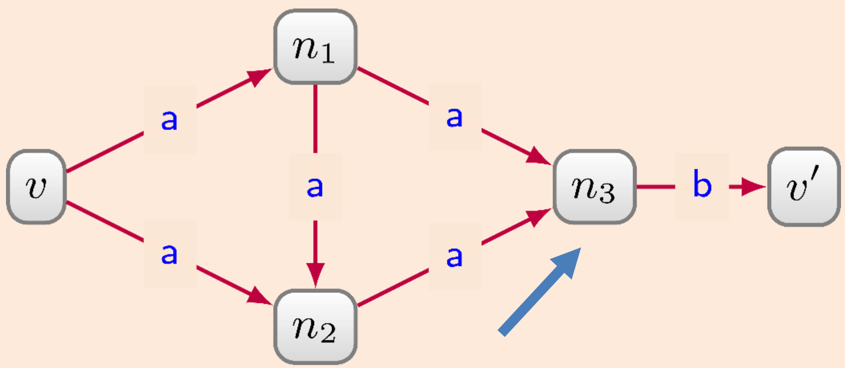


Let's see

Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    if $q == q_F$ **then**
        enumAllShortestPaths(current)
    for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
        if $(n', q', *, *) \in$ Visited **then**
            $(n', q', depth'$,prevList'$) \leftarrow$ Visited.get(n', q')
            if $depth + 1 == depth'$ **then**
                prevList'.add(current)
        else
            prevList.init()
            prevList.add(current)
            newState $\leftarrow (n', q', depth + 1$,prevList$)$
            Visited.push(newState)
            Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

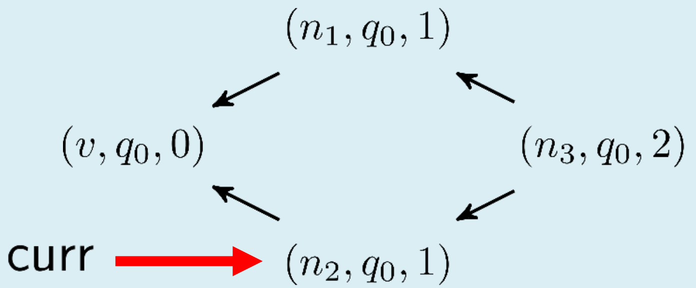




Open:

$(n_3, q_0, 2, \text{pl}_{n_3})$

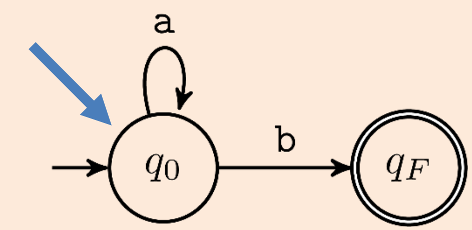
Visited:

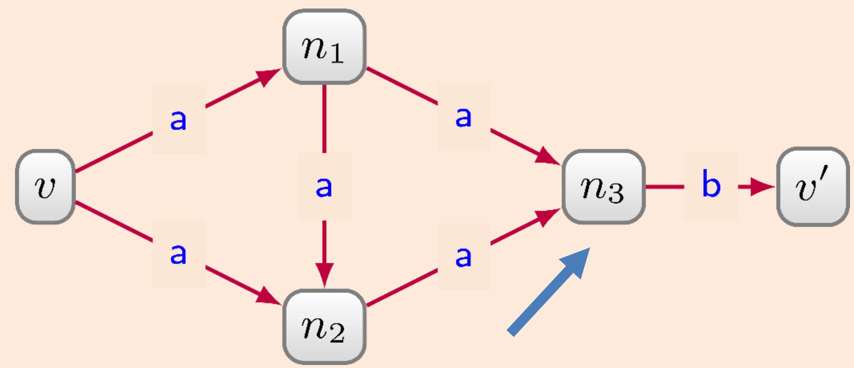


Let's see

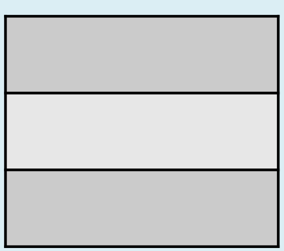
Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  if $q == q_F$ **then**
    enumAllShortestPaths(current)
  for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
    if $(n', q', *, *) \in$ Visited **then**
      $(n', q', depth'$,prevList'$)$ $\leftarrow$ Visited.get(n', q')
      if $depth + 1 == depth'$ **then**
        prevList'.add(current)
    else
      prevList.init()
      prevList.add(current)
      newState $\leftarrow (n', q', depth + 1$,prevList$)$
      Visited.push(newState)
      Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

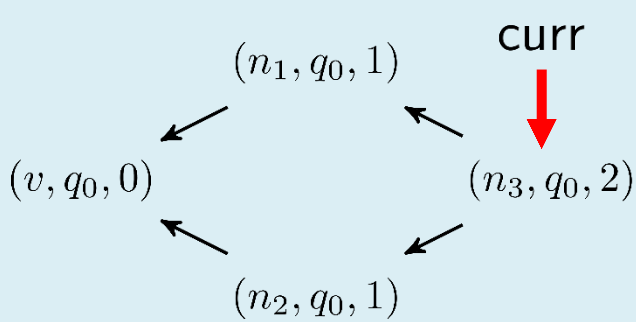




Open:



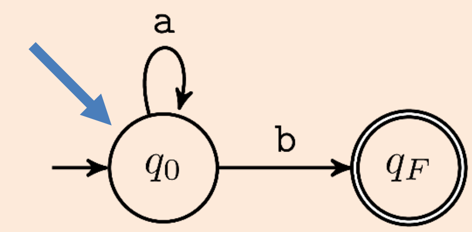
Visited:

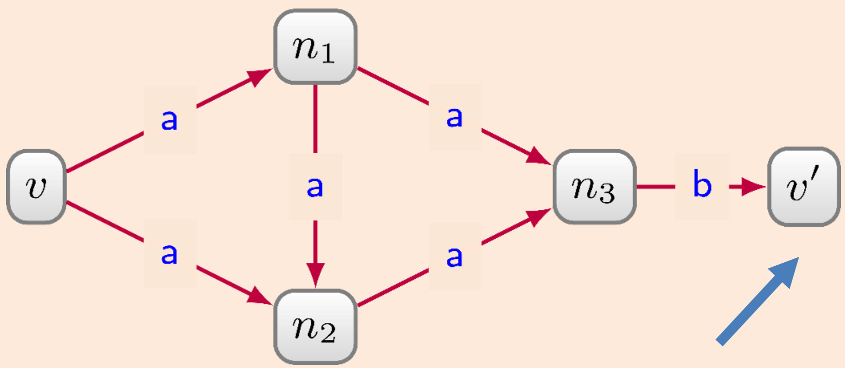


Let's see

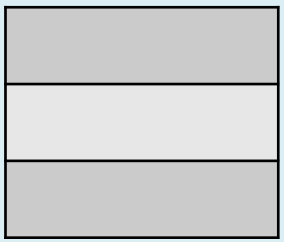
Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    if $q == q_F$ **then**
        enumAllShortestPaths(current)
    for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
        if $(n', q', *, *) \in$ Visited **then**
            $(n', q', depth'$,prevList'$) \leftarrow$ Visited.get(n', q')
            if $depth + 1 == depth'$ **then**
                prevList'.add(current)
        else
            prevList.init()
            prevList.add(current)
            newState $\leftarrow (n', q', depth + 1$,prevList$)$
            Visited.push(newState)
            Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

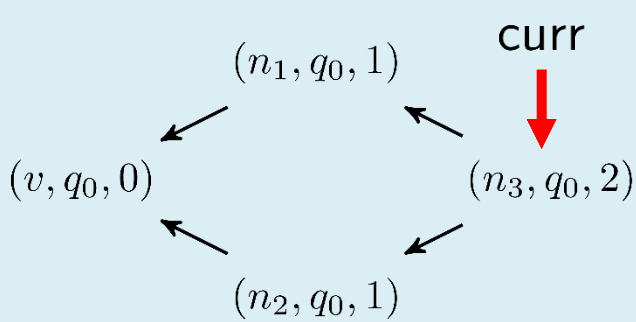




Open:



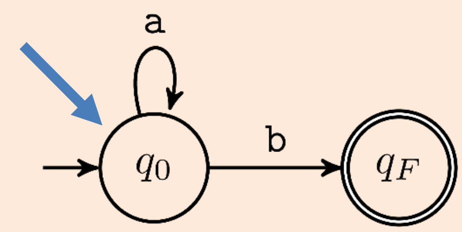
Visited:

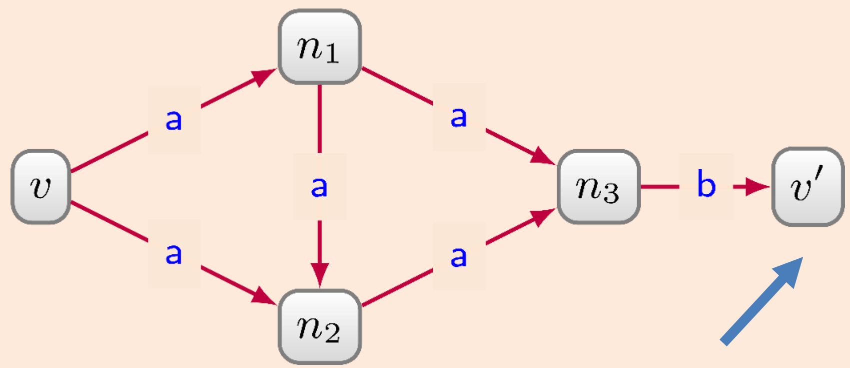


Let's see

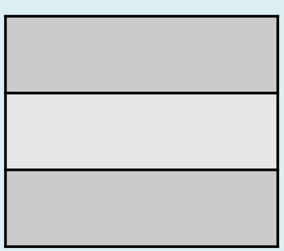
Open.init()
Visited.init()
startState $\leftarrow (v, q_0, 0, \perp)$
Visited.push(startState)
Open.push(startState)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    if $q == q_F$ **then**
        enumAllShortestPaths(current)
    for next $= (n', q') \in$ Neighbors(current, $G, \mathcal{A}$) **do**
        if $(n', q', *, *) \in$ Visited **then**
            $(n', q', depth'$,prevList') $\leftarrow$ Visited.get(n', q')
            if $depth + 1 == depth'$ **then**
                prevList'.add(current)
        else
            prevList.init()
            prevList.add(current)
            newState $\leftarrow (n', q', depth + 1$,prevList)
            Visited.push(newState)
            Open.push(newState)

ALL SHORTEST WALK $(v) = [a^*b] => (?x)$

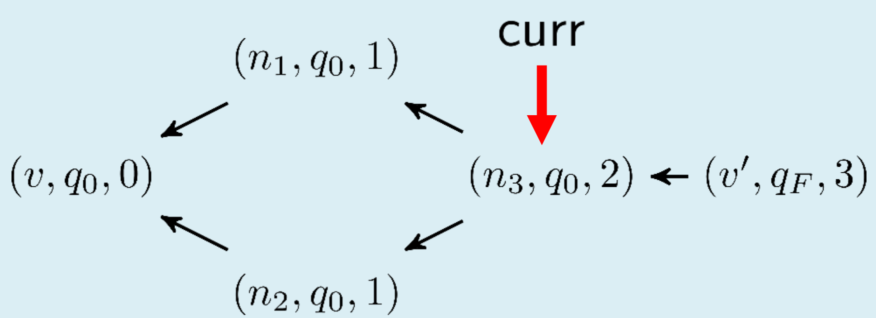




Open:



Visited:



ALL SHORTEST WALKS

$$\text{ALL SHORTEST WALK } (v) = [\texttt{regex}] => (?x)$$

Theorem. *Let G be a graph database and q the query:*

$$\text{ALL SHORTEST WALK } (v) = [\texttt{regex}] => (?x)$$

Computing the output of q over G can be done with $O(|\texttt{regex}| \times |G|)$ pre-processing and output-linear delay.

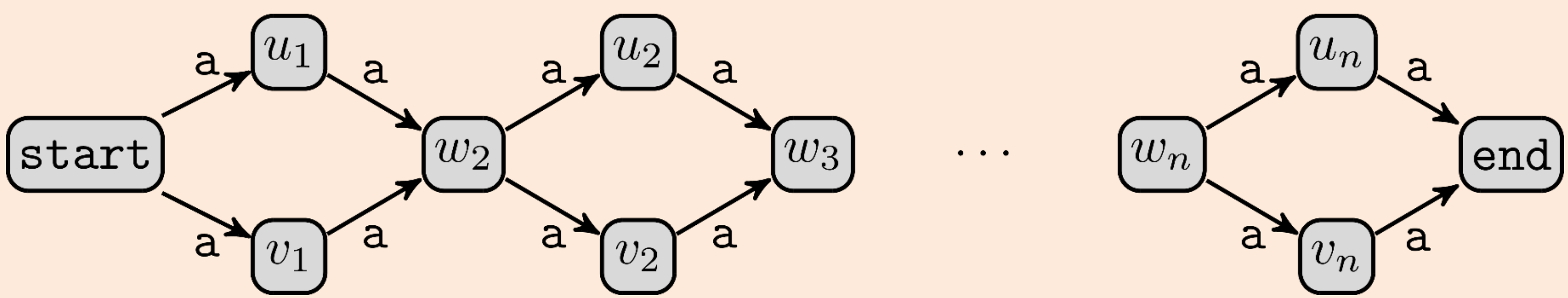
How come the complexity is the same as for ANY?

- Nothing extra is pushed onto the queue
- Sure, some additional edges are added to Visited
- But these were traversed in the standard BFS as well

Same as ANY

Yes, but you might have many more paths!

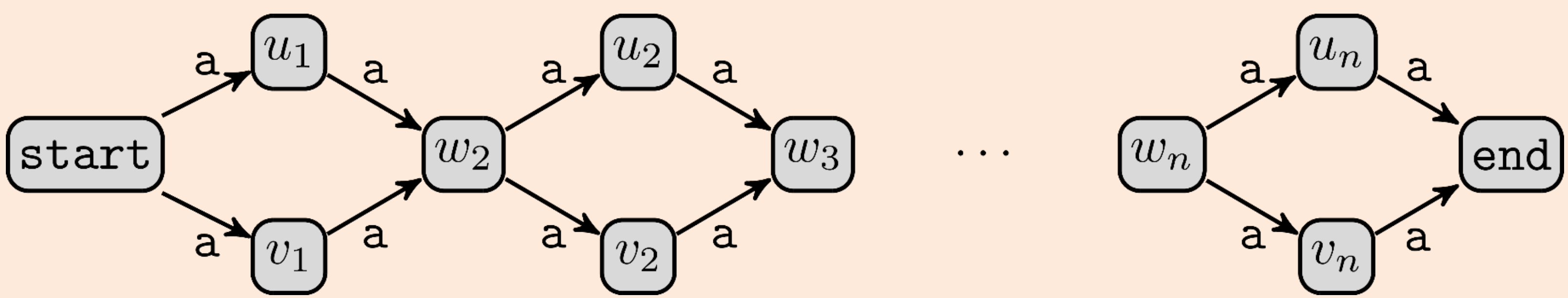
query = ALL SHORTEST WALK (start) = $[a^*]$ => (end)



Same as ANY

Yes, but you might have many more paths!

query = ALL SHORTEST WALK (start) $= [a^*] =>$ (end)

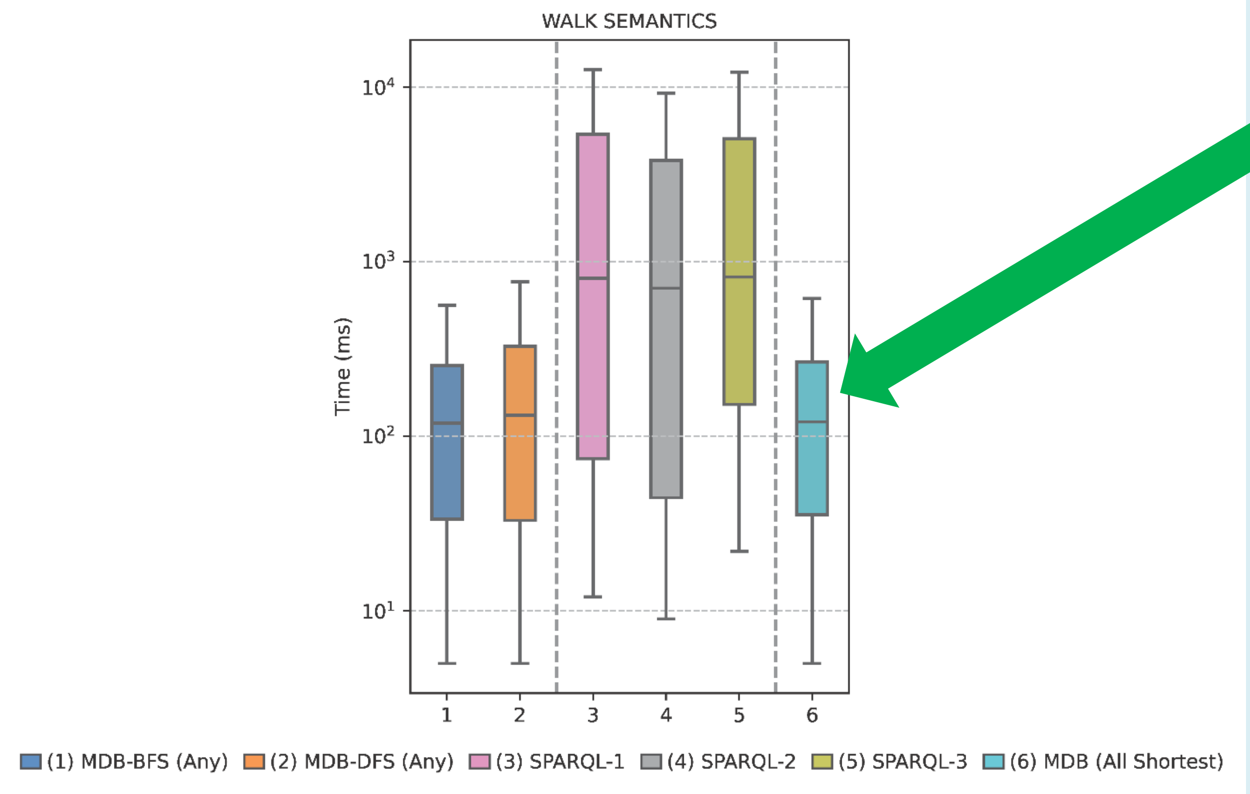


Exponentially more compact representation of the results

See [MNPRVV22] for details

Does this work in practice?

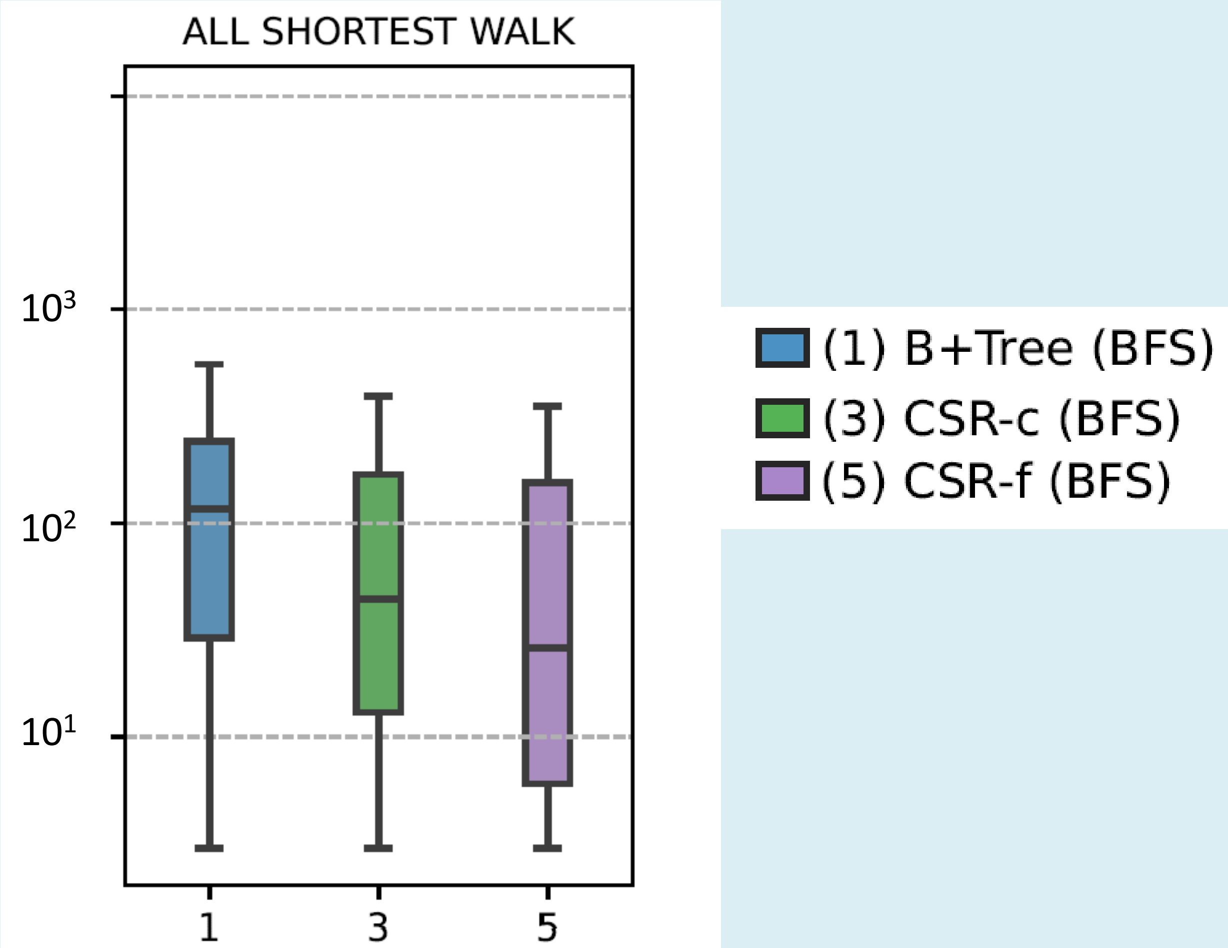
- Wikidata-based benchmark [WDBench]:
  - 1.25B edges (60000 edge labels)/300M nodes
  - 659 (non-bot) user defined queries ([MKGGB18])
  - (100,000 limit – some queries have >10M results, 1min timeout)



See [MDB, FMRV23]

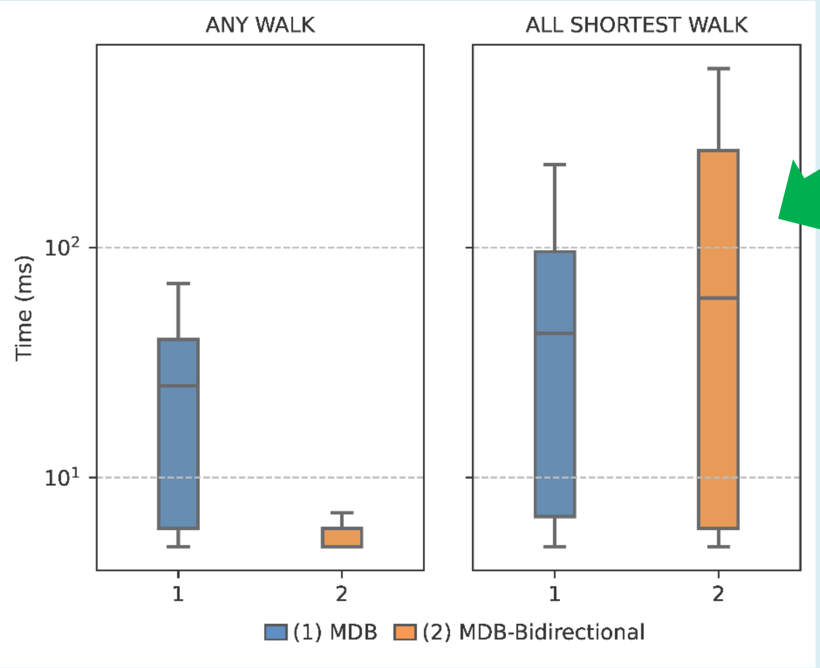
Considerations 1

- How does CSR perform?



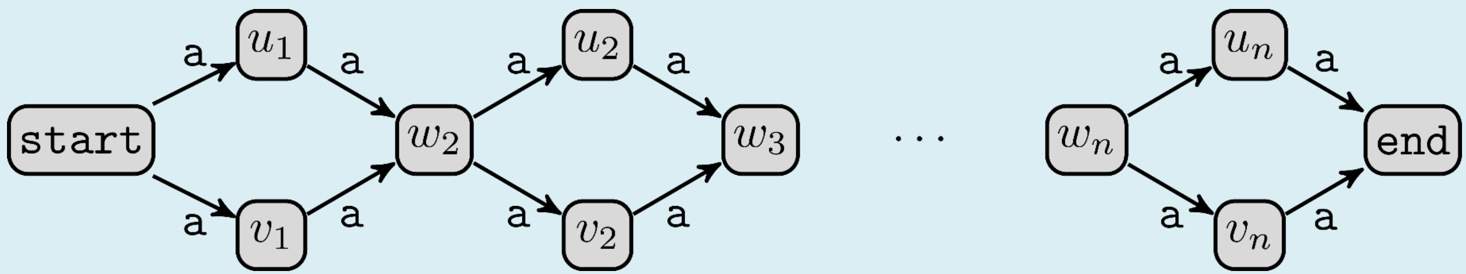
Considerations 2

- All assumptions on automaton can be lifted [DFM23]
- Same CSR/B+tree discussion applies
- For fixed (src,tgt) two-way approach has issues

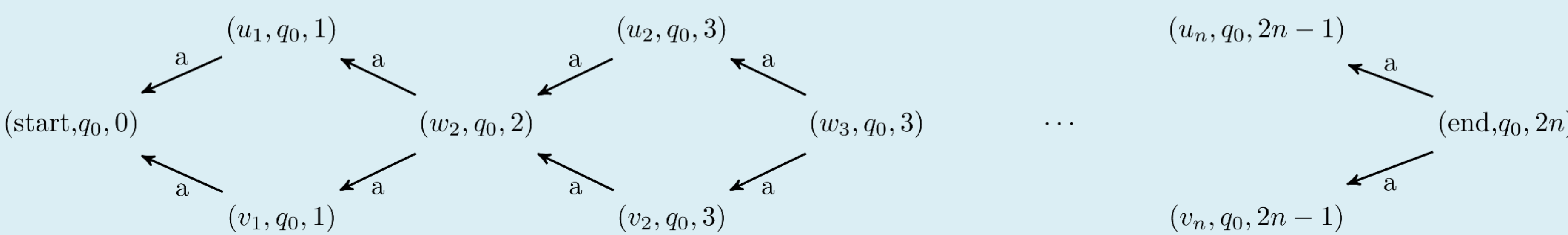


Considerations 3

query = ALL SHORTEST WALK (start) = $[a^*]$ => (end)



- The compressed representation (PMR) really shines:



Simple paths and Trails
(bonus slides)

Simple paths

$$\text{ANY SIMPLE } (v) = [\texttt{regex}] => (?x)$$

Theorem. *Let G be a graph database and q the query:*

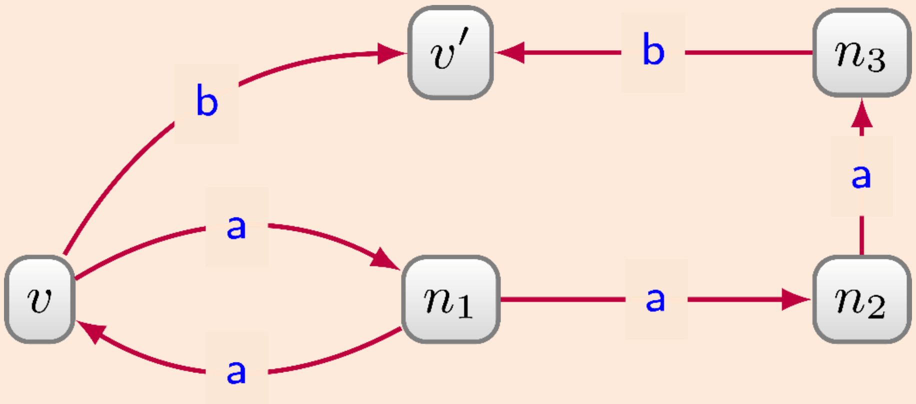
$$\text{ANY SIMPLE } (v) = [\texttt{regex}] => (?x)$$

Chekcing whether q has a single answer over G is NP-complete.

What is the problem here?

Simple paths – when to stop?

$$\text{ANY SIMPLE } (v) = [\texttt{a}^+\texttt{b}] => (?x)$$



Shortest: v → n1 → v → v’

Simple: v → n1 → n2 → n3 → v’

for $\texttt{a}^+\texttt{b}$

Simple paths – the idea

The algorithm is quite stupid (as any NP-hard one):

- Iterate over all possible paths in the product graph
- If the path in the original graph is simple continue
- If the path is not simple stop extending it

Why does this terminate?

- Max path length = $|V|$
- So $|V|^{|V|}$ candidates

ANY SIMPLE

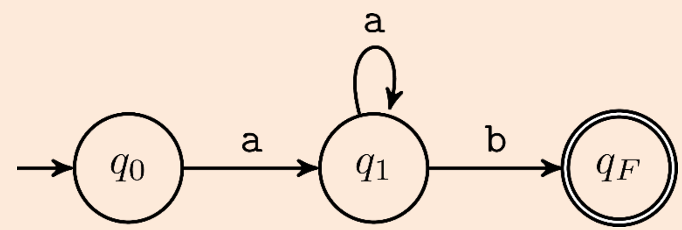
Algorithm 1 Algorithm for $?p$ = ANY SIMPLE ($(v) = [\texttt{regex}] \Rightarrow (?x)$)

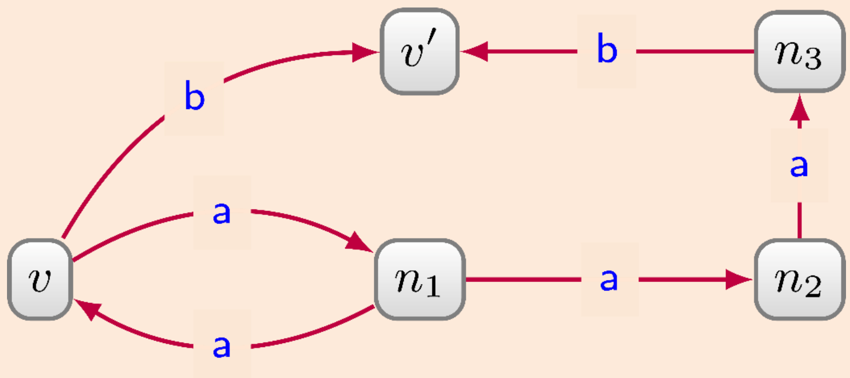
```
function AnySimple(G, q)
    𝒜 ← Automaton(regex)                          ▷ q_0 initial, q_F final
    Open.init()                                   ▷ Queue of searchStates
    Visited.init()                                ▷ Set coding visited paths in G
    ReachedFinal.init()                           ▷ Discovered solutions
    start ← (v, q_0, ⊥)
    Visited.push(start)
    Open.push(start)
    while !Open.isEmpty() do
        current ← Open.pop()                      ▷ current = (n, q,prev)
        for next = (n′, q′) ∈ Neighbors(current, G, 𝒜) do
            if isSimple(current, n′) then         ▷ Extending with n′ is OK
                new ← (n′, q′,current)
                Visited.push(new)
                Open.push(new)
                if q′ == q_F then                 ▷ Solution is found
                    if n′ ∉ ReachedFinal then     ▷ For the first time
                        ReachedFinal.add(n′)
                        getPath(new)
```

Let's see

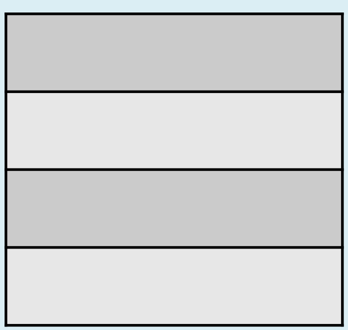
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q'$,,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^{+}\mathtt{b}] => (?x)$





Open:

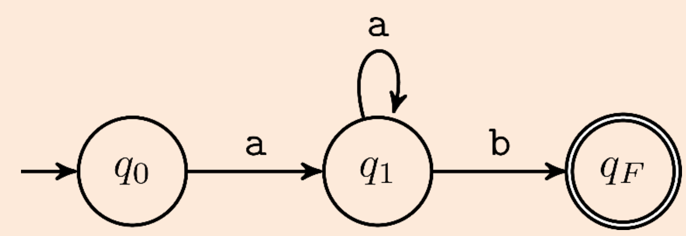


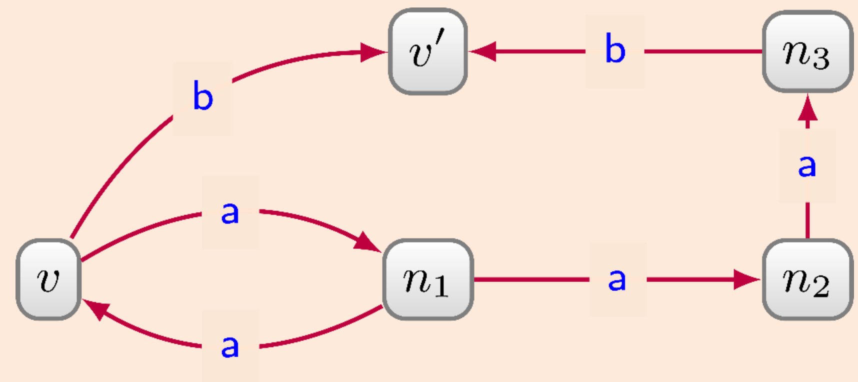
Visited:

Let's see

start $\leftarrow (v, q_0, \bot)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q',$,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] \Rightarrow (?x)$





Open:

$(v, q_0, \bot)$

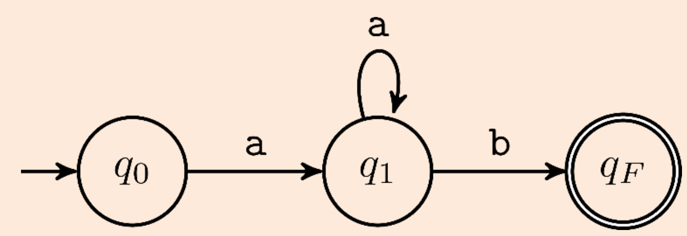
Visited:

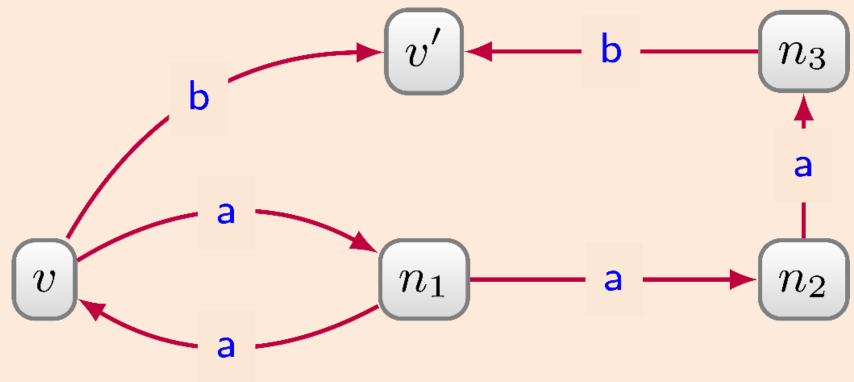
(v, q_0)

Let's see

```
start ← (v, q0, ⊥)
Visited.push(start)
Open.push(start)
while !Open.isEmpty() do
    current ← Open.pop()
    for (n′, q′) ∈ Neighbors(current) do
        if isSimple(current, n′) then
            new ← (n′, q′,,current)
            Visited.push(new)
            Open.push(new)
            if q′ == qF then
                if n′ ∉ ReachedFinal then
                    ReachedFinal.add(n′)
                    getPath(new)
```

ANY SIMPLE $(v) = [\mathtt{a}^{+}\mathtt{b}] => (?x)$





Open:

$(v, q_0, \perp)$

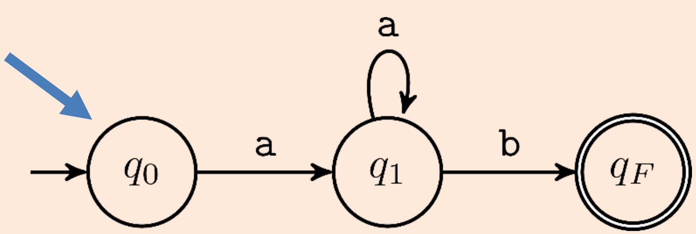
Visited:

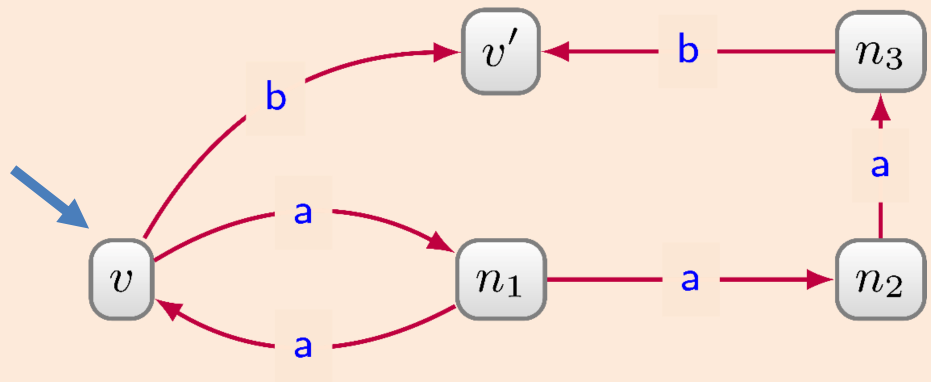
(v, q_0)

Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q',$,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

Visited:

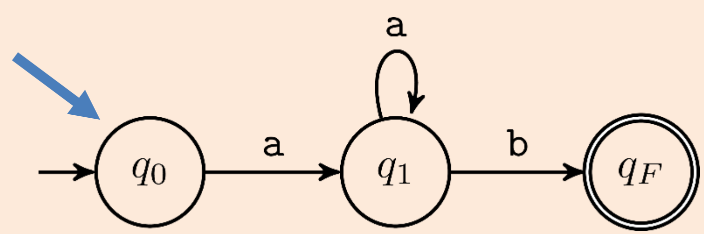
(v, q_0)

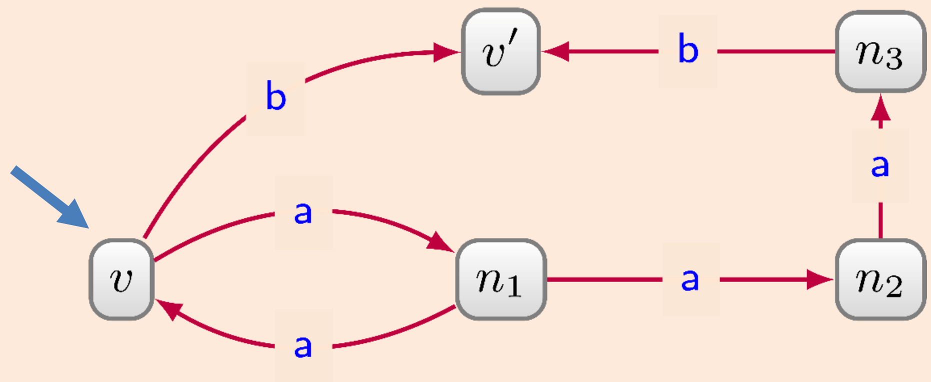
curr

Let's see

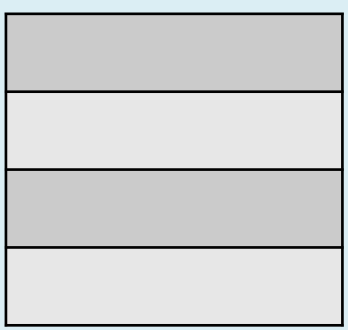
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q',$,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$

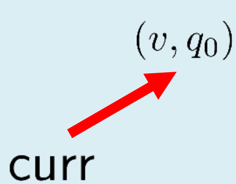




Open:



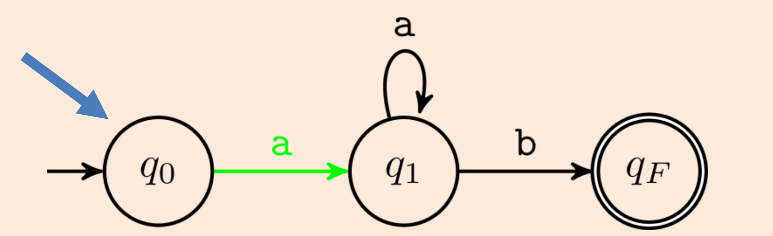
Visited:

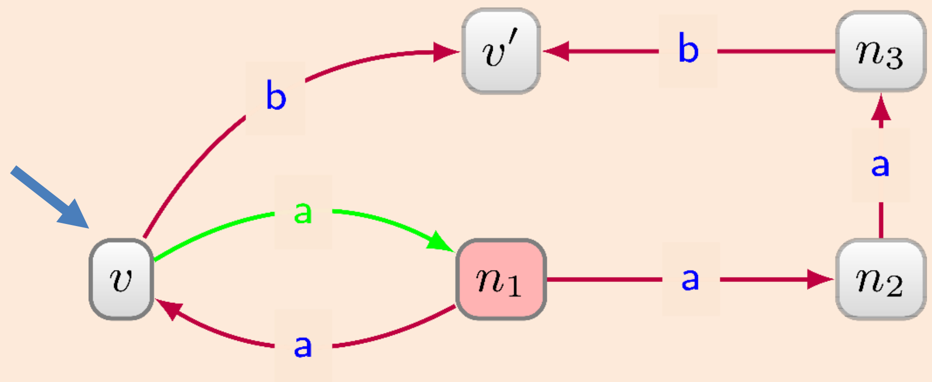


Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q',$,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

Visited:

(v, q_0)

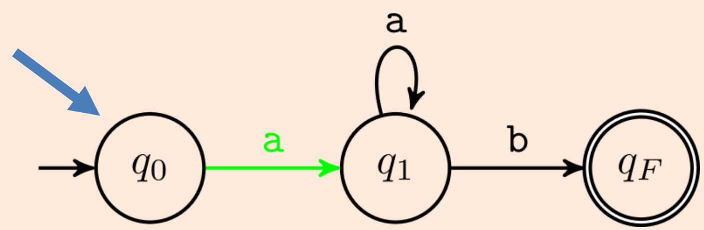
curr

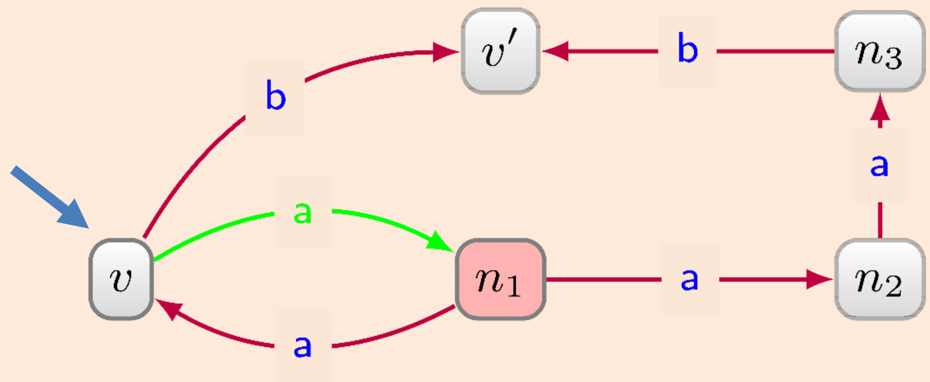
Let's see

In G

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q',$,current)
      Visited.push(new)
      Open.push(new)
    if $q' == q_F$ **then**
      if $n' \notin$ ReachedFinal **then**
        ReachedFinal.add(n')
        getPath(new)

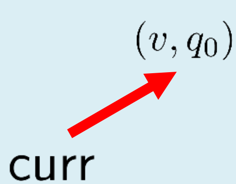
ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

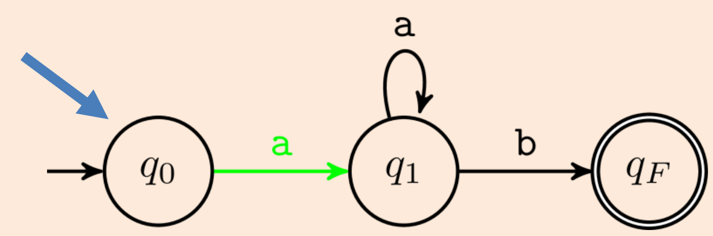
Visited:

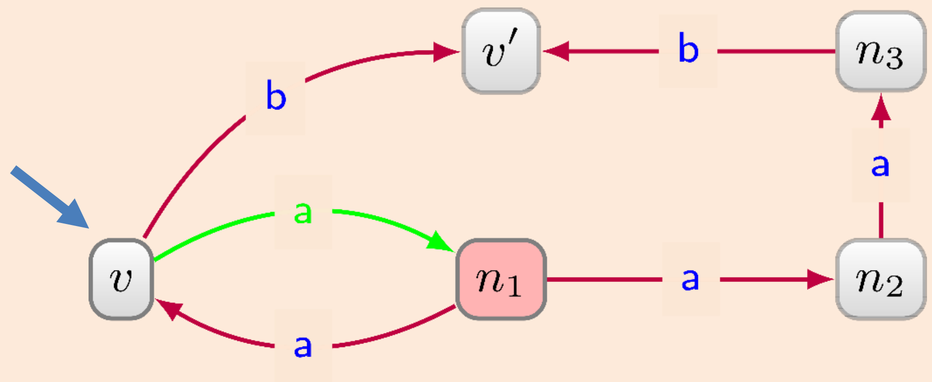


Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q',$,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

$(n_1, q_1,$ prev$)$

Visited:

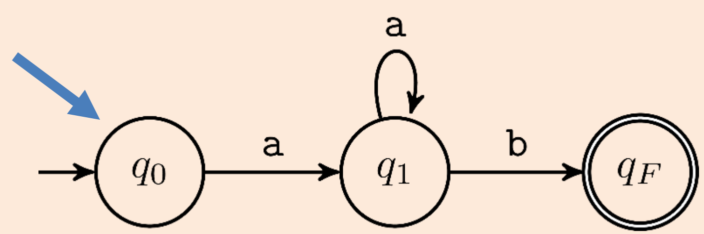
$(v, q_0) \longleftarrow (n_1, q_1)$

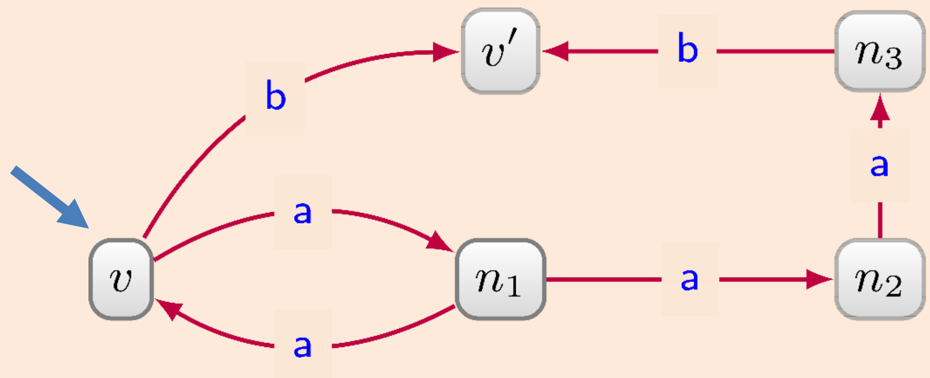
curr

Let's see

start $\leftarrow (v, q_0, \bot)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q'$,,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

$(n_1, q_1,$ prev$)$

Visited:

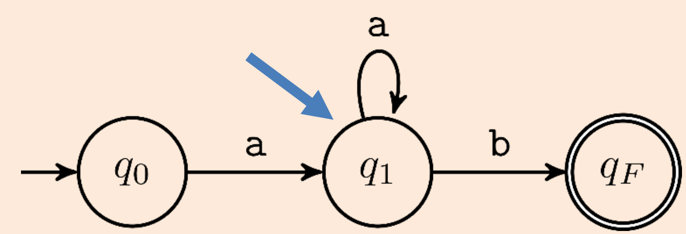
$(v, q_0) \longleftarrow (n_1, q_1)$

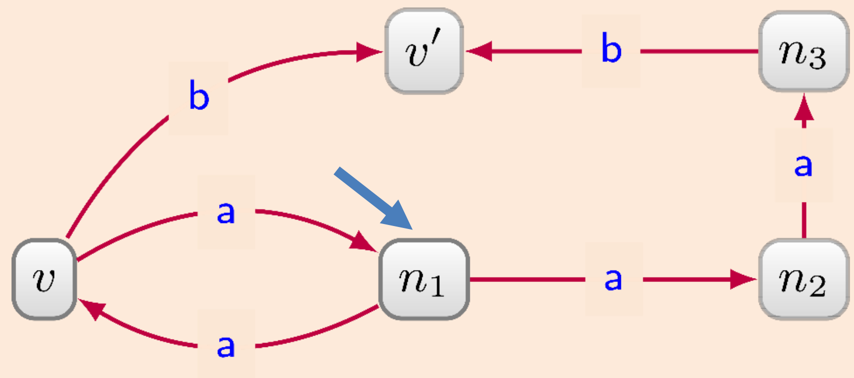
curr

Let's see

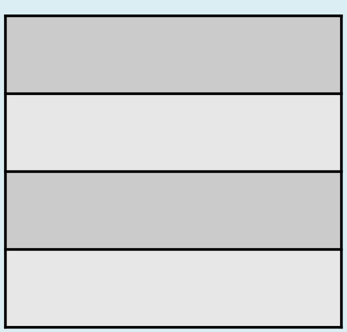
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q'$,,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\texttt{a}^+\texttt{b}] => (?x)$





Open:



Visited:

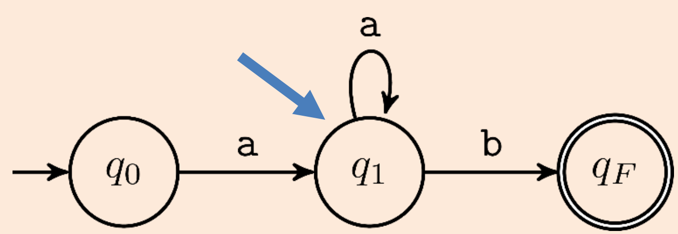
$(v, q_0) \longleftarrow (n_1, q_1)$

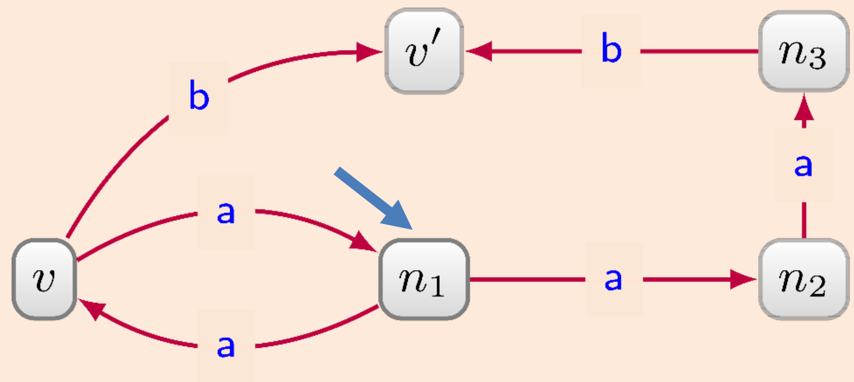
curr

Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q'$,,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

Visited:

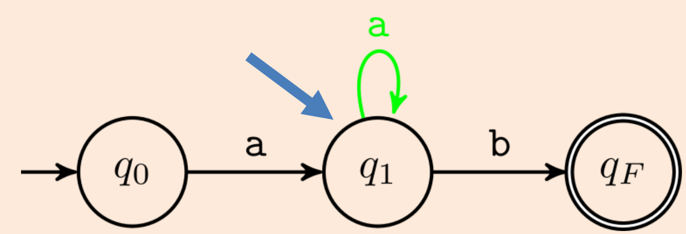
$(v, q_0) \longleftarrow (n_1, q_1)$

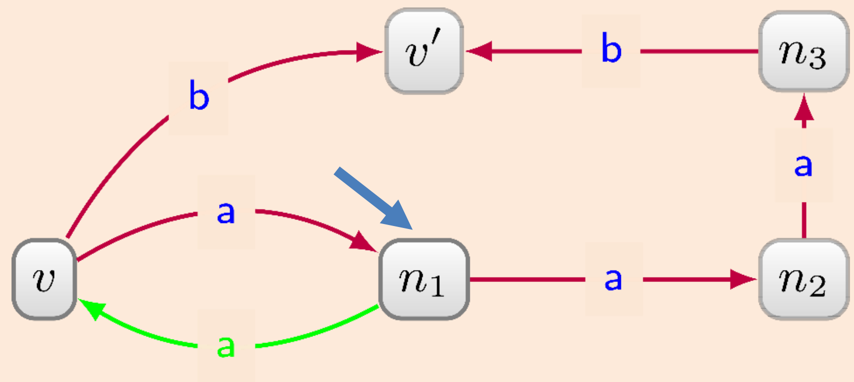
curr

Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q',$,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

Visited:

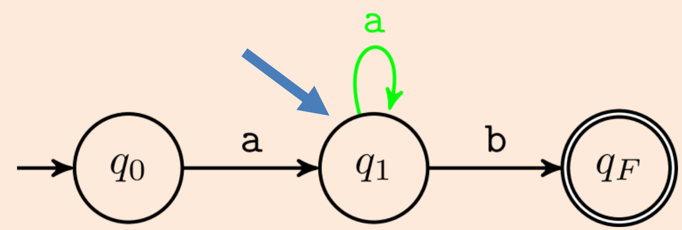
$(v, q_0) \longleftarrow (n_1, q_1)$

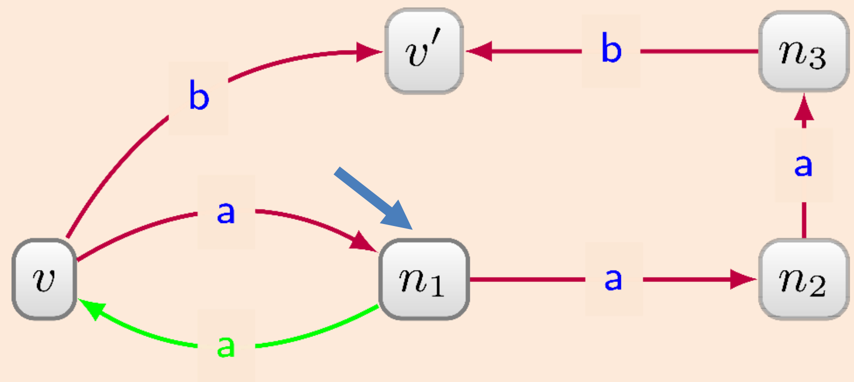
curr

Let's see

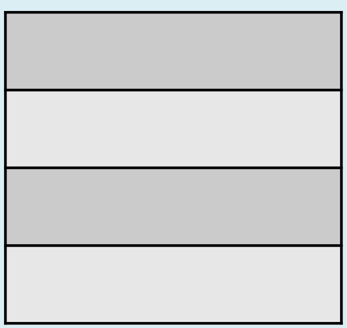
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q'$,,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] \Rightarrow (?x)$





Open:



Visited:

$(v, q_0) \longleftarrow (n_1, q_1)$

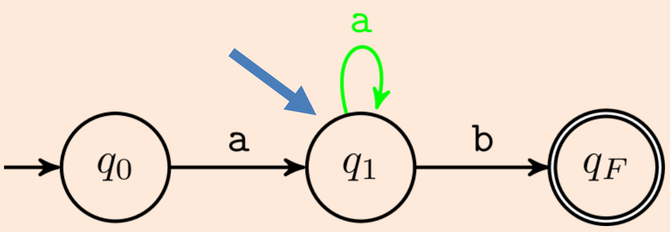
curr

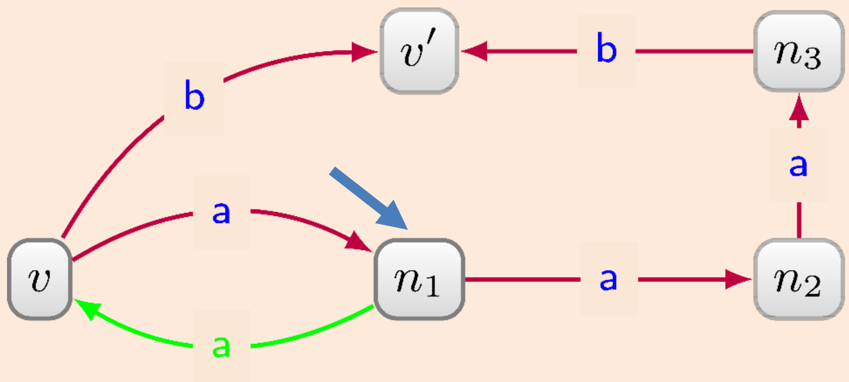
Let's see

In G!!!

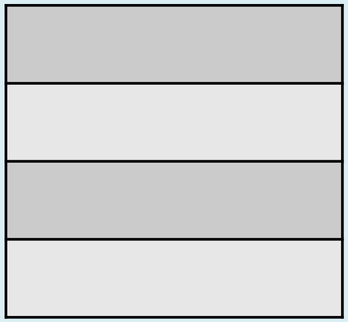
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q'$,,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] \Rightarrow (?x)$

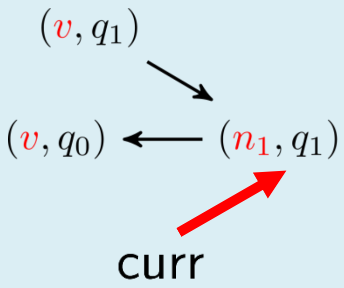




Open:



Visited:

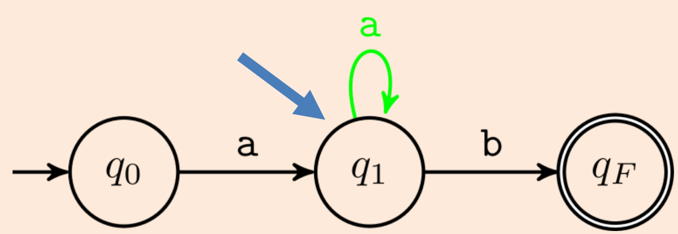


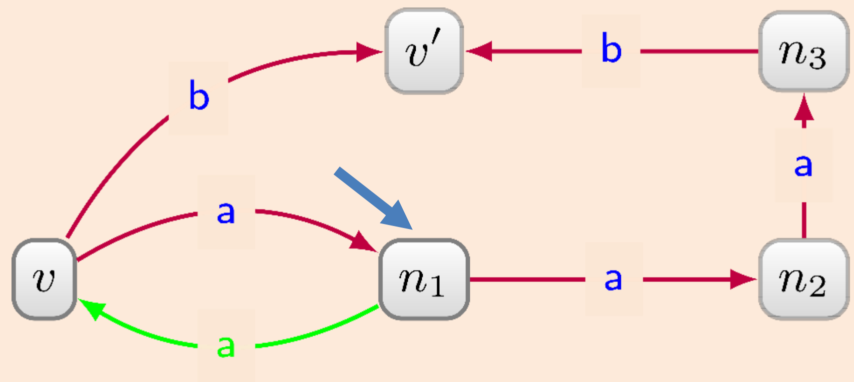
Let's see

```
start ← (v, q0, ⊥)
Visited.push(start)
Open.push(start)
while !Open.isEmpty() do
    current ← Open.pop()
    for (n′, q′) ∈ Neighbors(current) do
        if isSimple(current, n′) then
            new ← (n′, q′,,current)
            Visited.push(new)
            Open.push(new)
            if q′ == qF then
                if n′ ∉ ReachedFinal then
                    ReachedFinal.add(n′)
                    getPath(new)
```

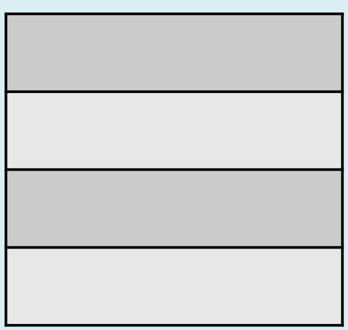
In G!!!

ANY SIMPLE $(v) = [\mathtt{a}^{+}\mathtt{b}] => (?x)$

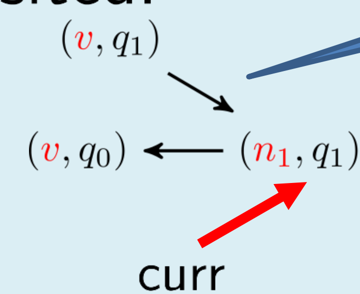




Open:



Visited:

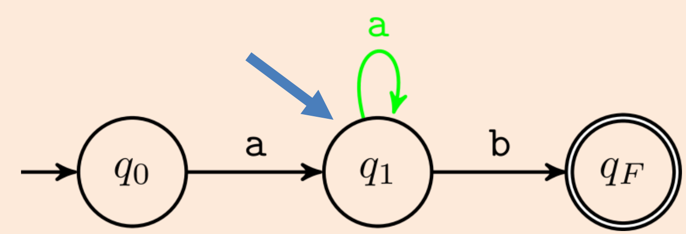


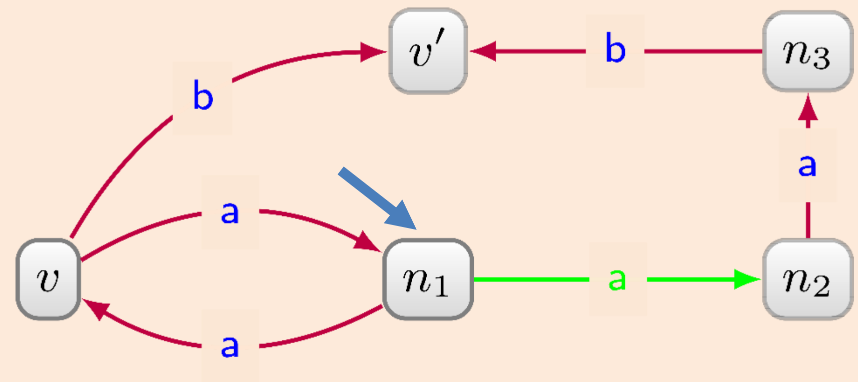
Simple path in the product

Let's see

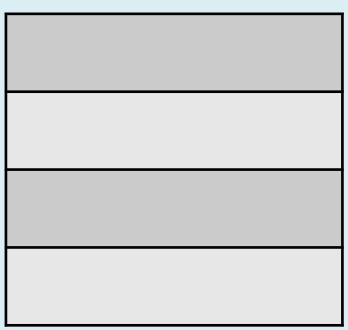
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q',$,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:



Visited:

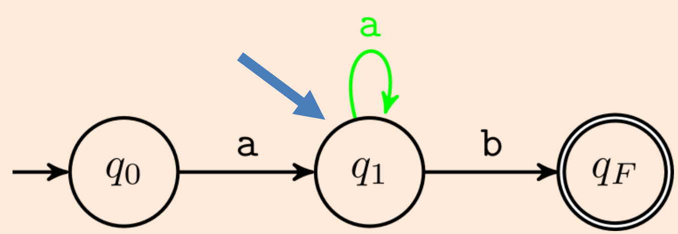
$(v, q_0) \longleftarrow (n_1, q_1)$

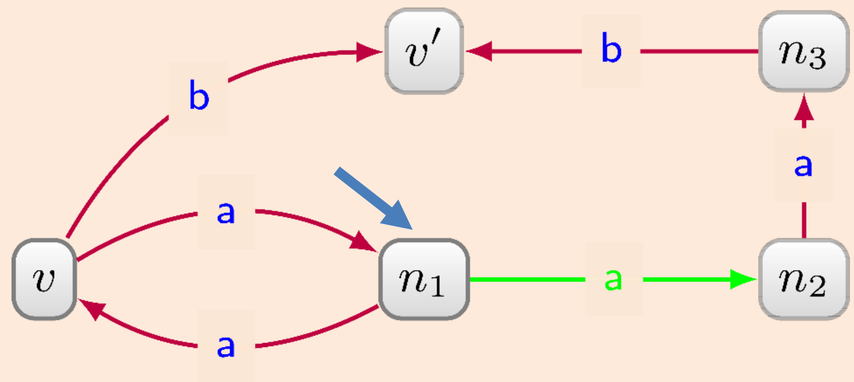
curr

Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q',$,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$

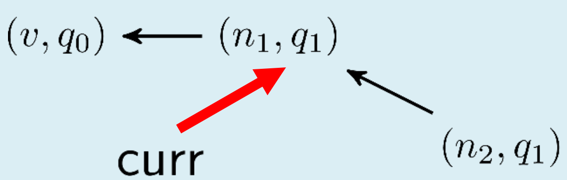




Open:

$(n_2, q_1,$ prev$)$

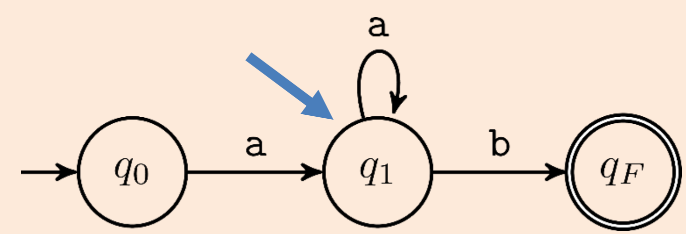
Visited:

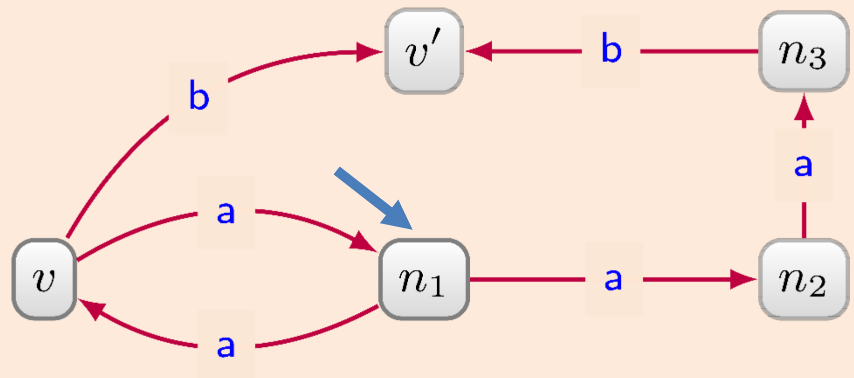


Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q'$,,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^{+}\mathtt{b}] => (?x)$

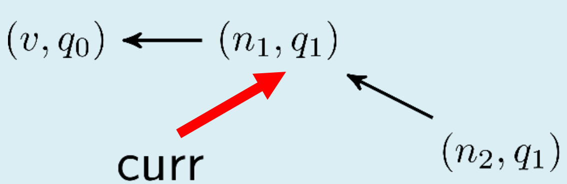




Open:

(n_2, q_1, prev)

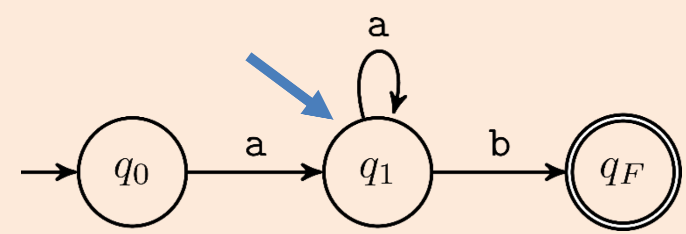
Visited:

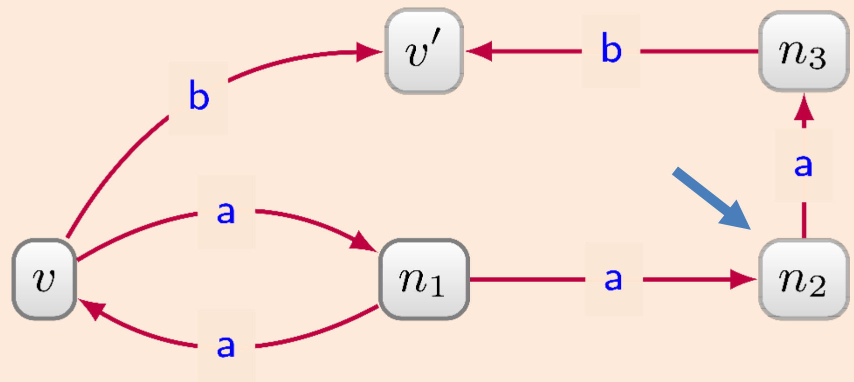


Let's see

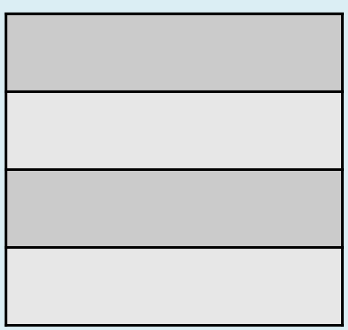
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q'$,,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$

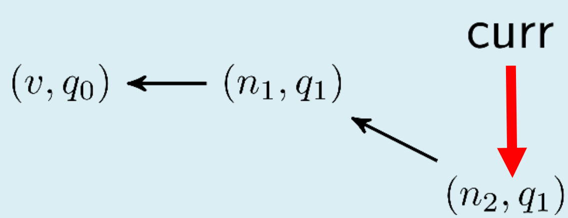




Open:



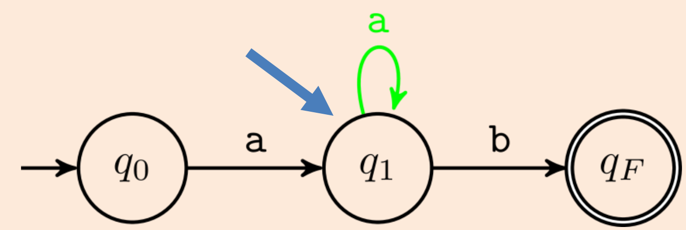
Visited:

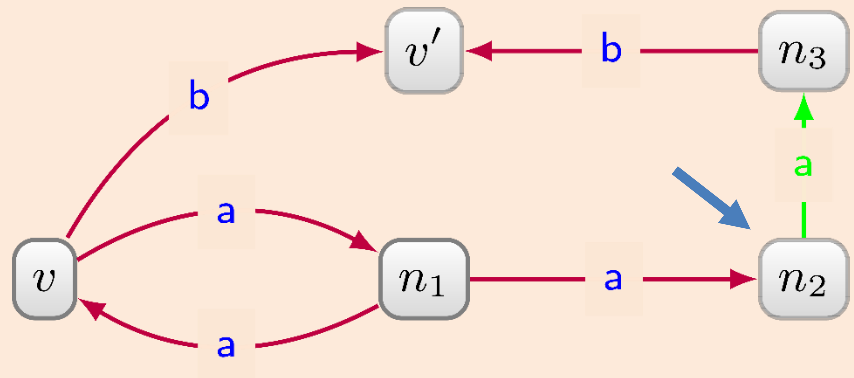


Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q',$,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$

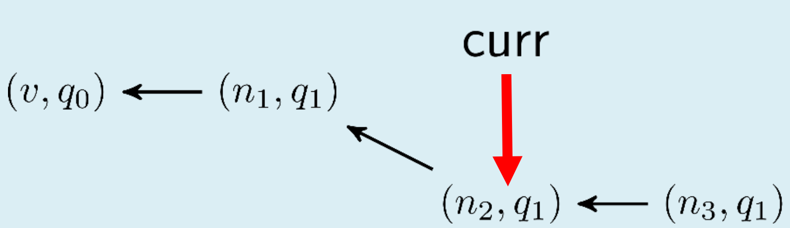




Open:

$(n_3, q_1,$ prev$)$

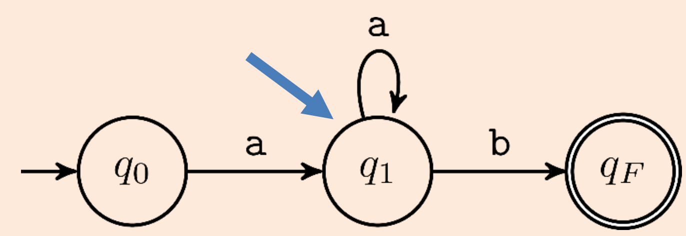
Visited:

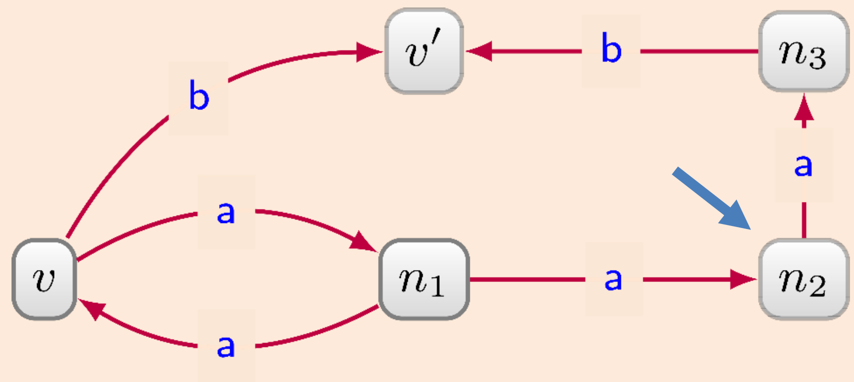


Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q'$,,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$

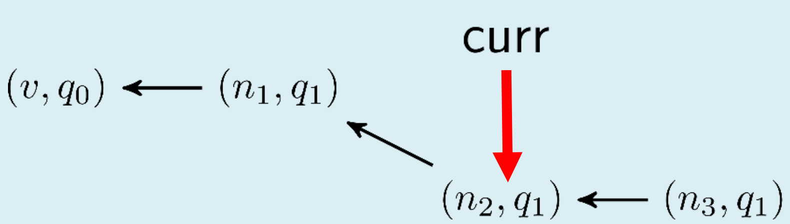




Open:

$(n_3, q_1$, prev)

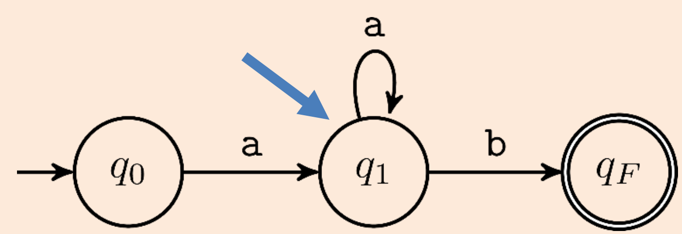
Visited:

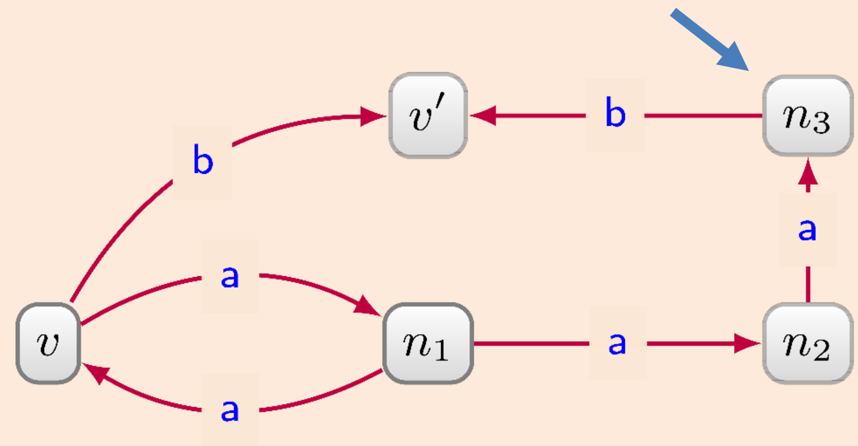


Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q'$,,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

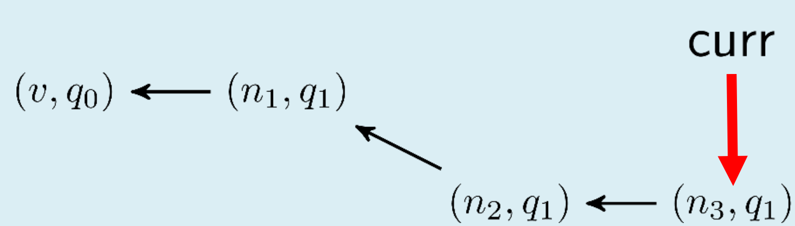
ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

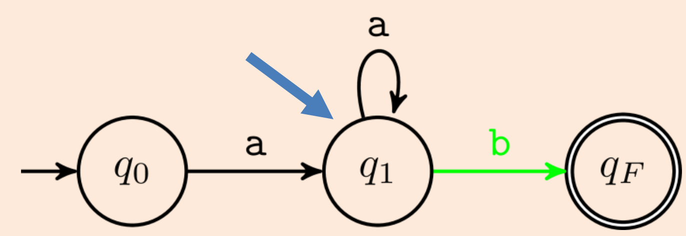
Visited:

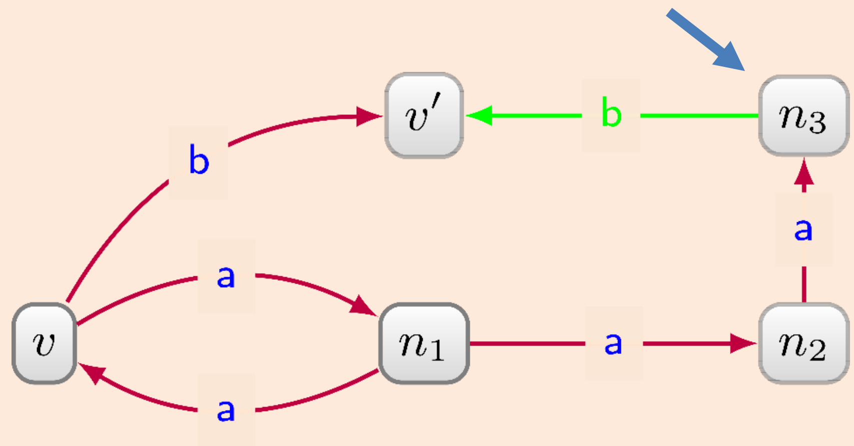


Let's see

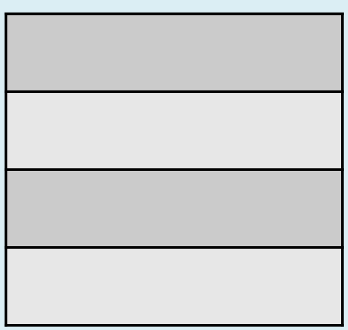
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
current $\leftarrow$ Open.pop()
for $(n', q') \in$ Neighbors(current) **do**
if isSimple(current, n') **then**
new $\leftarrow (n', q',$,current)
Visited.push(new)
Open.push(new)
if $q' == q_F$ **then**
if $n' \notin$ ReachedFinal **then**
ReachedFinal.add(n')
getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$

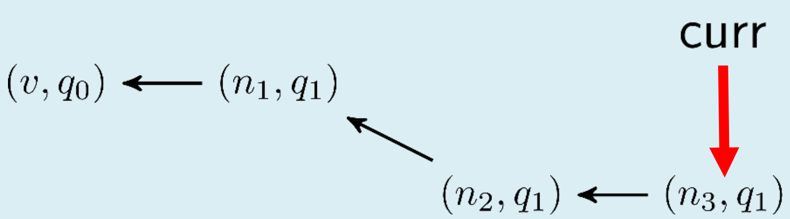




Open:



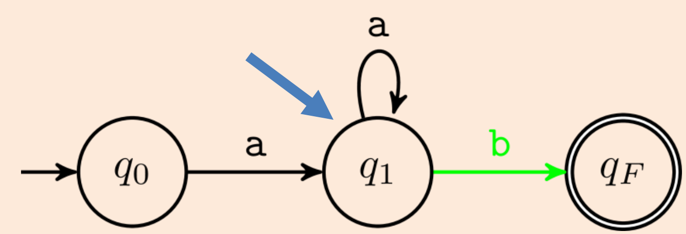
Visited:

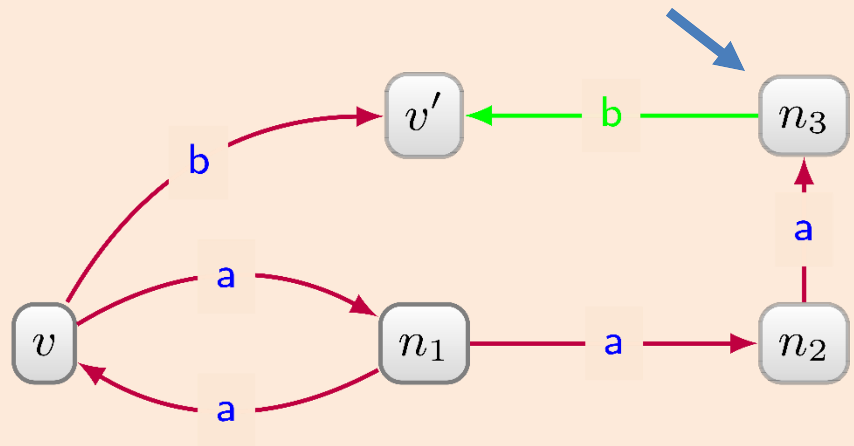


Let's see

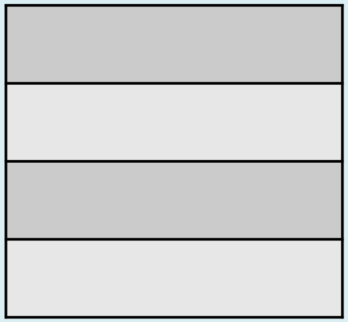
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q',$,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\texttt{a}^+\texttt{b}] => (?x)$





Open:



Visited:

curr

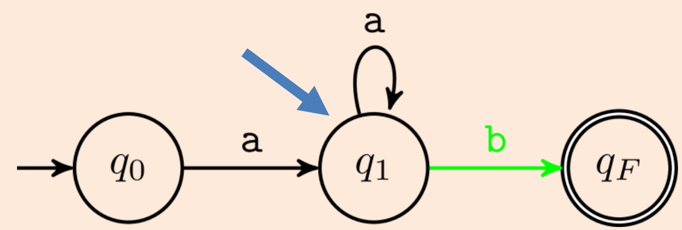
$(v, q_0) \leftarrow (n_1, q_1)$

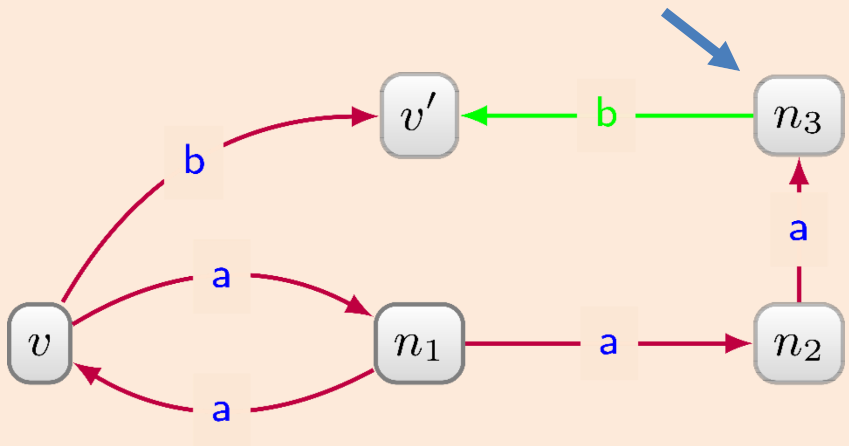
$(n_2, q_1) \leftarrow (n_3, q_1) \leftarrow (v', q_F)$

Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q'$,,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

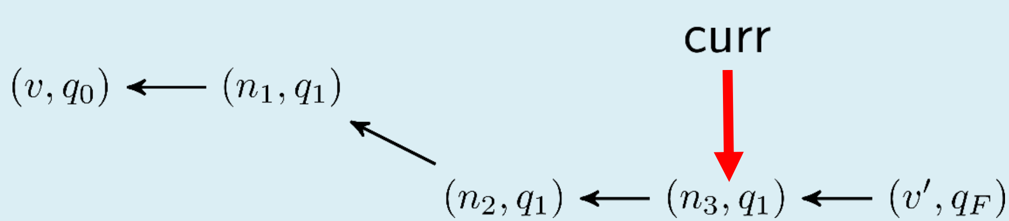
ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$





Open:

Visited:

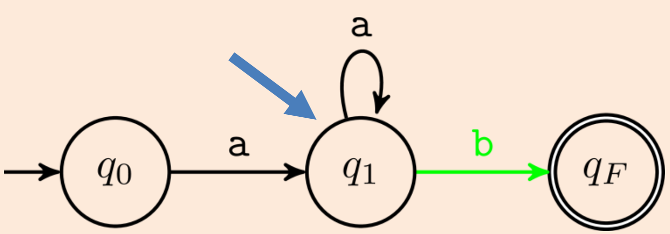


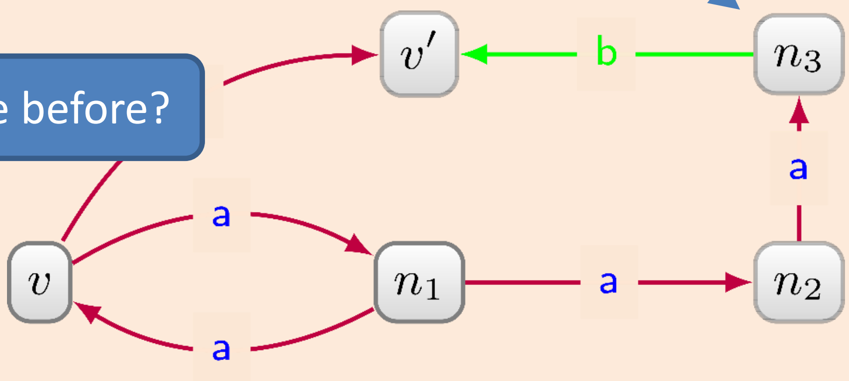
Let’s see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q',$,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                $\boxed{\textbf{if } n' \notin \text{ReachedFinal } \textbf{then}}$
                    ReachedFinal.add(n')
                    getPath(new)

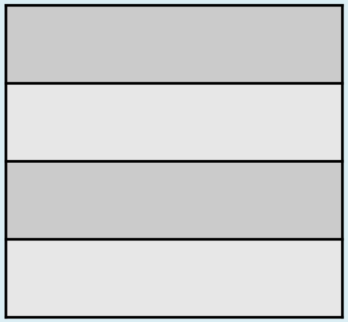
Was I here before?

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$

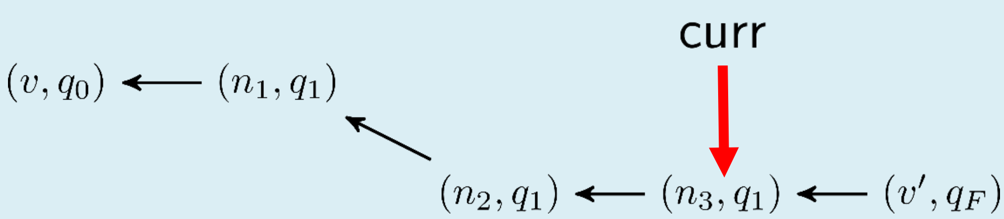




Open:



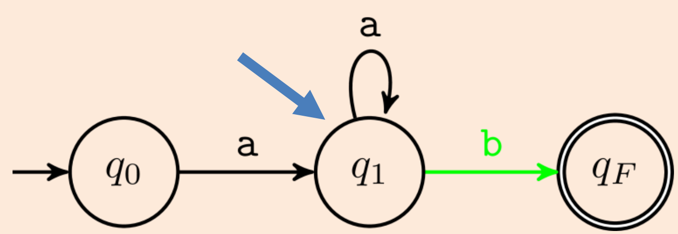
Visited:

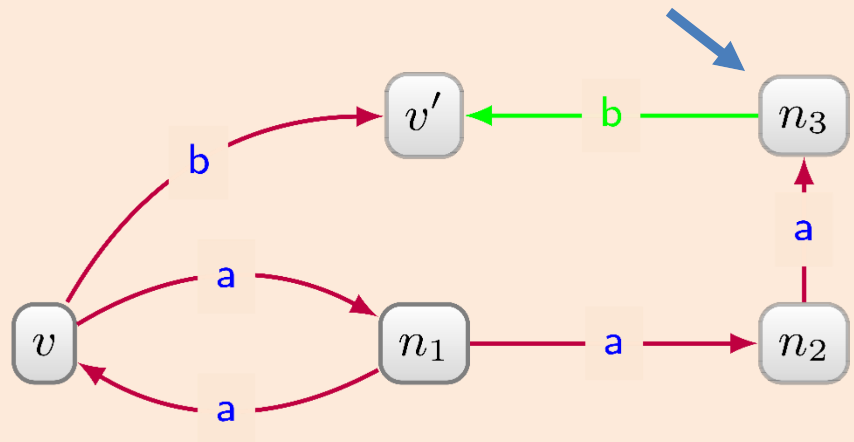


Let's see

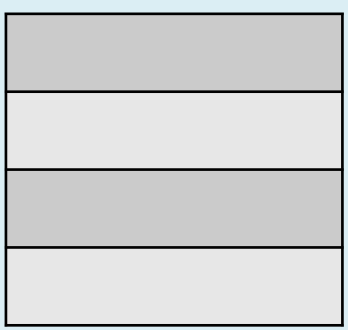
start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop()
    for $(n', q') \in$ Neighbors(current) **do**
        if isSimple(current, n') **then**
            new $\leftarrow (n', q'$,,current)
            Visited.push(new)
            Open.push(new)
            if $q' == q_F$ **then**
                if $n' \notin$ ReachedFinal **then**
                    ReachedFinal.add(n')
                    getPath(new)

ANY SIMPLE $(v) = [\mathtt{a}^+\mathtt{b}] => (?x)$



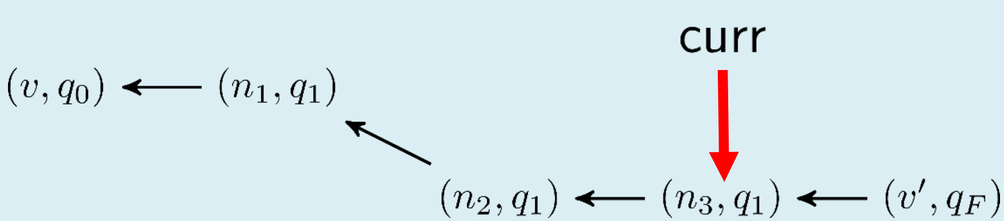


Open:



Visited:

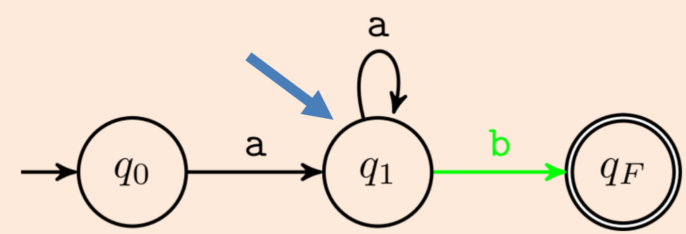
ReachedFinal: $\{v'\}$

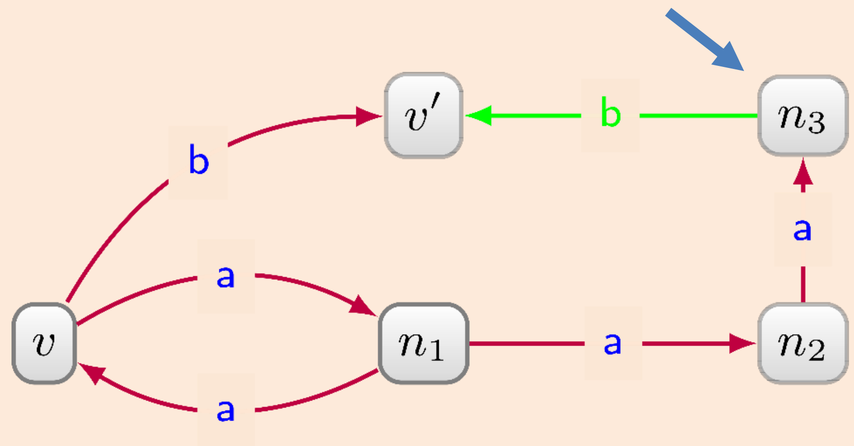


Let's see

start $\leftarrow (v, q_0, \perp)$
Visited.push(start)
Open.push(start)
while !Open.isEmpty() **do**
  current $\leftarrow$ Open.pop()
  for $(n', q') \in$ Neighbors(current) **do**
    if isSimple(current, n') **then**
      new $\leftarrow (n', q',$,current)
      Visited.push(new)
      Open.push(new)
      if $q' == q_F$ **then**
        if $n' \notin$ ReachedFinal **then**
          ReachedFinal.add(n')
          getPath(new)

ANY SIMPLE $(v) = [\texttt{a}^+\texttt{b}] => (?x)$





Open:

Visited:

ReachedFinal: $\{v'\}$

curr

$(v, q_0) \longleftarrow (n_1, q_1)$

$(n_2, q_1) \longleftarrow (n_3, q_1) \longleftarrow (v', q_F)$

And this guy?

$$\text{SIMPLE } (v) = [\texttt{regex}] => (?x)$$

Algorithm 1 Algorithm for $?p = \text{SIMPLE } ((v) = [\texttt{regex}] => (?x))$

function AnySimple(G, q)
    $\mathcal{A} \leftarrow$ Automaton($\texttt{regex}$) ▷ q_0 initial, q_F final
    Open.init() ▷ Queue of searchStates
    Visited.init() ▷ Set coding visited paths in G
    start $\leftarrow (v, q_0, \perp)$
    Visited.push(start)
    Open.push(start)
    while !Open.isEmpty() **do**
        current $\leftarrow$ Open.pop() ▷ current $= (n, q,$prev$)$
        for next $= (n', q') \in$ Neighbors(current$, G, \mathcal{A}$) **do**
            if isSimple(current, n') **then** ▷ Extending with n' is OK
                new $\leftarrow (n', q',$current$)$
                Visited.push(new)
                Open.push(new)
                if $q' == q_F$ **then** ▷ Solution is found
                    getPath(new)

And this guy?

Needs to be unambiguous

SIMPLE $(v) = [\texttt{regex}] => (?x)$

Algorithm 1 Algorithm for $?p$ = SIMPLE ($(v) = [\texttt{regex}] => (?x)$)

function ANYSIMPLE(G, q)
  $\mathcal{A} \leftarrow$ Automaton($\texttt{regex}$) ▷ q_0 initial, q_F final
  Open.init() ▷ Queue of searchStates
  Visited.init() ▷ Set coding visited paths in G
  start $\leftarrow (v, q_0, \perp)$
  Visited.push(start)
  Open.push(start)
  while !Open.isEmpty() **do**
    current $\leftarrow$ Open.pop() ▷ current $= (n, q,$prev$)$
    for next $= (n', q') \in$ Neighbors(current$, G, \mathcal{A}$) **do**
      if isSimple(current, n') **then** ▷ Extending with n' is OK
        new $\leftarrow (n', q',$current$)$
        Visited.push(new)
        Open.push(new)
        if $q' == q_F$ **then** ▷ Solution is found
          getPath(new)

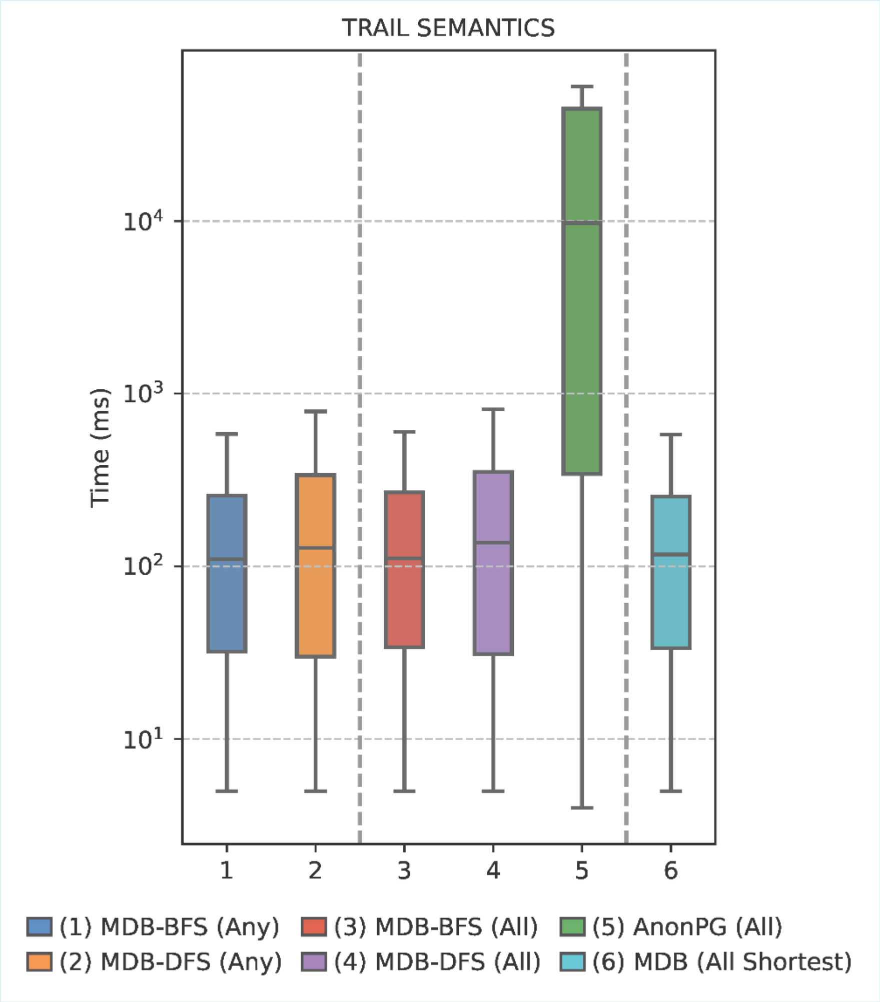
In general

Easily extended to:

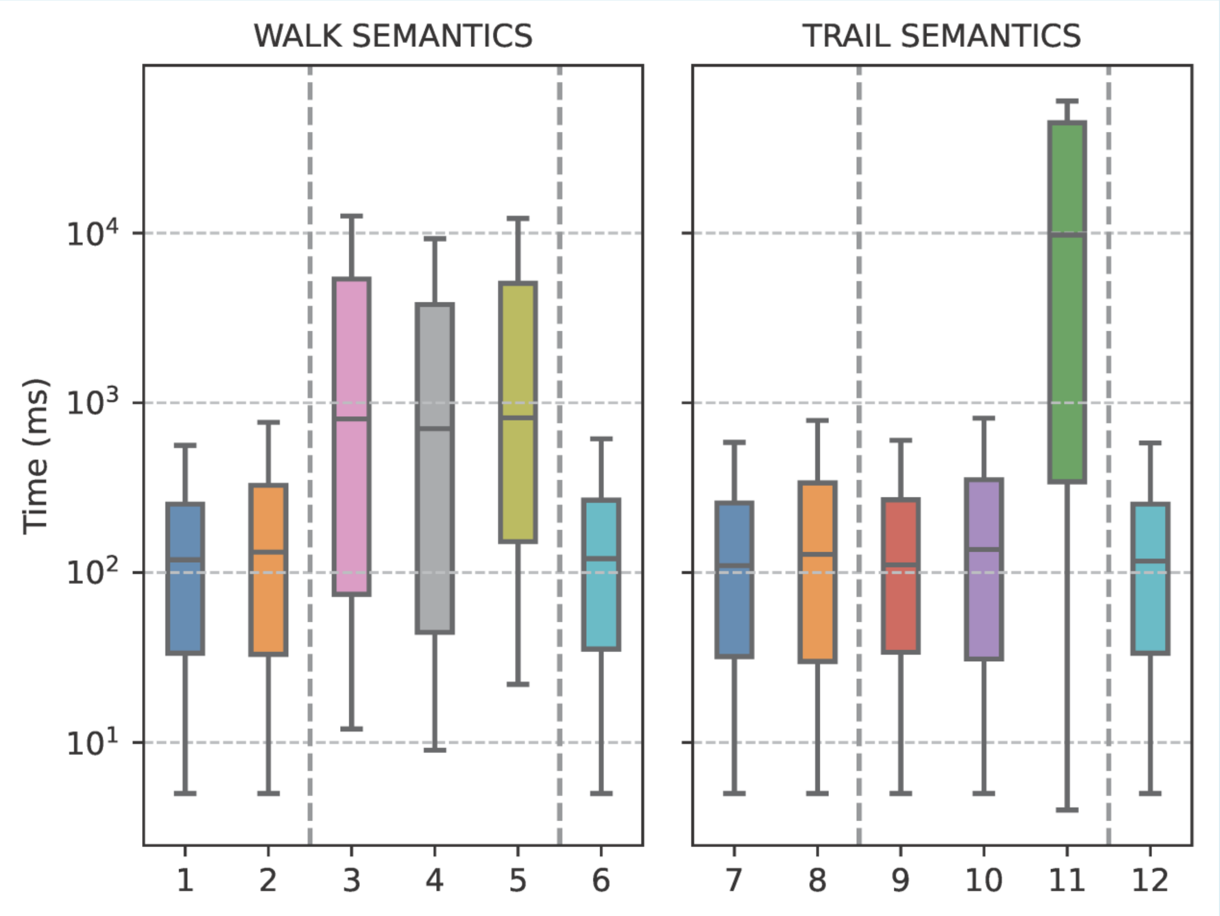
- ANY SHORTEST SIMPLE (we already did this)
- ALL SHORTEST SIMPLE (a bit of work)
- TRAIL

Basically, all the same algorithm

Does this work in practice?



Does this work in practice?



Almost the same as "tractable" semantics!

TLDR; on path queries

Product graph construction [MW95]:

- Robust enough to support GQL requirements
  - We just use a different graph exploration method
- Can be coupled with different graph storage model
  - We tested for B+trees and CSR
- Compact representation of query results (when possible)
  - Exponential savings for ALL SHORTEST
- Pipelined execution easy to achieve
  - Pause/resume as soon as one path is found
- Works on real-world graphs
  - At least on Wikidata with user defined queries

Basically not a bad way to go!

What next?

- We only discussed a single path query on its own
  - CRPQ evaluation is still quite unexplored
- No attribute values considered in our queries
  - Reasoning on those can be algorithmically challenging [LMV16]
- Aggregation over paths is highly contentious
  - Easily becomes undecidable [GPC23]
- GQL is still adding new features
  - Group variables introduce some intersting challenges [GQLDigest23]

Lots of interesting problems to solve!

Part 4: MillenniumDB

IMFD Chile

- The graph engine we built:

MillenniumDB

Key highlights of MillenniumDB

- RDF/SPARQL & Property Graphs/GQL
  - Inside of the same engine
  - SPARQL path queries extended with GQL-inspired features

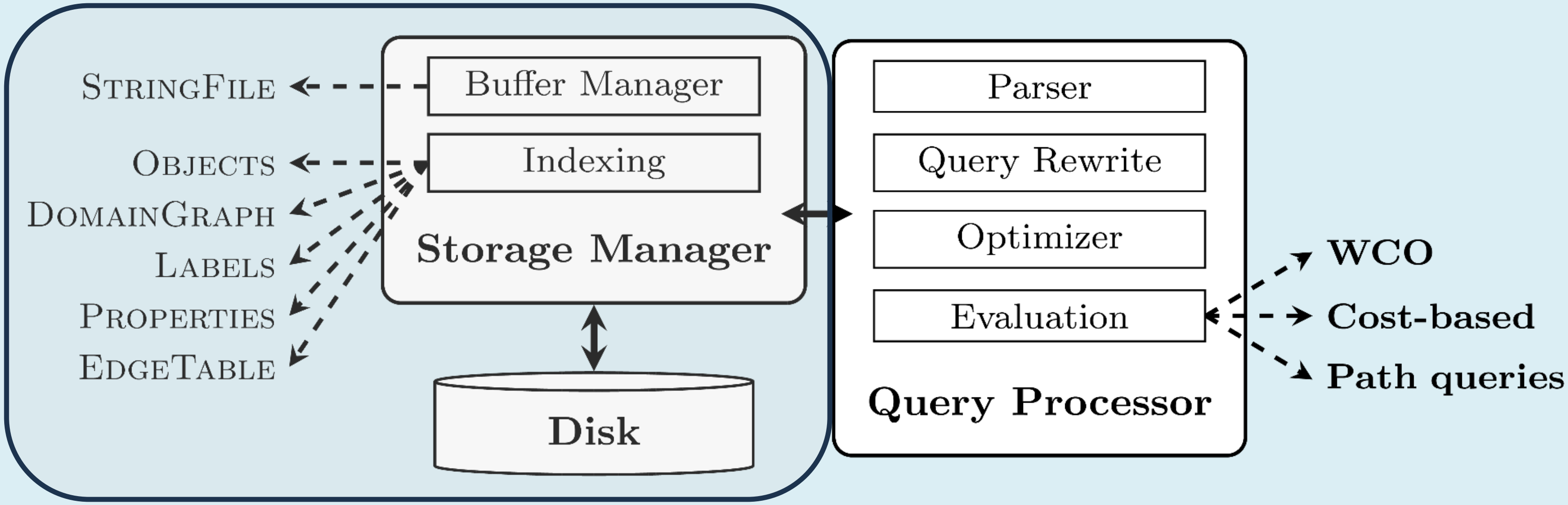
- Classical database pipeline
  - Quasi-relational

- Focus on support for public query endpoints
  - MVCC-based concurrency control
  - Readers always go through
  - Cental update mechanism

Implementation details

- Worst-case optimal join processing
  - Excelent join performance
  - Storage/update heavy
- Path queries
  - First engine supporting all GQL path queries
  - Builds on the theoretical concept of enumeration algorithms
- B+tree storage
  - Multiple permutations supporting wco-joins
  - Leaf compression (Wikidata shows huge savings)
  - Also support for CSR for path queries

Architecture of MillenniumDB



RDF **Triples**(subject, predicate, object)

PGs
Connections(src, label, tgt, <u>eId</u>)
Labels(objectId, label)
Properties(<u>objectId</u>, <u>key</u>, value)

Try it yourself

https://github.com/MillenniumDB/MillenniumDB

Try it yourself

https://wikidata.imfd.cl

https://mdb.imfd.cl/path_finder

https://bibkg.imfd.cl/

https://telarkg.imfd.cl/

References

References

[AABHRV17] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, Domagoj Vrgoč: Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50(5): 68:1-68:40 (2017)

[B13] Pablo Barceló Baeza: Querying graph databases. PODS 2013: 175-188

[BDRV17] Jorge Baier, Dietrich Daroch, Juan L. Reutter, Domagoj Vrgoč: Evaluating navigational RDF queries over the Web. HyperText 2017.

[MNPRVV22] Wim Martens, Matthias Niewerth, Tina Popp, Carlos Rojas, Stijn Vansummeren, Domagoj Vrgoč: Representing Paths in Graph Database Pattern Matching. CoRR abs/2207.13541 (2022)

[MW95] Alberto O. Mendelzon, Peter T. Wood: Finding Regular Simple Paths in Graph Databases. SIAM J. Comput. 24(6): 1235-1258 (1995)

[V22] Domagoj Vrgoč: Evaluating regular path queries under the all-shortest paths semantics. CoRR abs/2204.11137 (2022)

[DFM23] Claire David, Nadime Francis, Victor Marsault: Distinct Shortest Walk Enumeration for RPQs. CoRR abs/2312.05505 (2023)

References

[FMRV23] Benjamín Farías, Wim Martens, Carlos Rojas, Domagoj Vrgoč: Evaluating Regular Path Queries in GQL and SQL/PGQ: How Far Can The Classical Algorithms Take Us? CoRR abs/2306.02194 (2023)

[WDBench] Renzo Angles, Carlos Buil Aranda, Aidan Hogan, Carlos Rojas, Domagoj Vrgoč: WDBench: A Wikidata Graph Query Benchmark. ISWC 2022

[MKGGB18] Malyshev, S., Krotzsch, M., Gonzalez, L., Gonsior, J., Bielefeldt, A.: Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In: ISWC 2018

[MDB] Domagoj Vrgoč, Carlos Rojas, Renzo Angles, Marcelo Arenas, Diego Arroyuelo, Carlos Buil-Aranda, Aidan Hogan, Gonzalo Navarro, Cristian Riveros, Juan Romero: MillenniumDB: An Open-Source Graph Database System. Data Intell. 5(3): 560-610 (2023) https://github.com/MillenniumDB

[XVG19] Chengshuo Xu, Keval Vora, Rajiv Gupta: PnP: Pruning and Prediction for Point-To-Point Iterative Graph Analytics. APLOS 2019

References

[GPC23] Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, Domagoj Vrgoč: GPC: A Pattern Calculus for Property Graphs. PODS 2023

[GQLDigest23] Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, Domagoj Vrgoč: A Researcher's Digest of GQL. ICDT 2023

[LMV16] Leonid Libkin, Wim Martens, Domagoj Vrgoč: Querying Graphs with Data. J. ACM 63(2) (2016)

[Multi22] Renzo Angles, Aidan Hogan, Ora Lassila, Carlos Rojas, Daniel Schwabe, Pedro A. Szekely, Domagoj Vrgoč: Multilayer graphs: a unified data model for graph databases. GRADES-NDA@SIGMOD 2022

[Reification] Daniel Hernández, Aidan Hogan, Markus Krötzsch: Reifying RDF: What Works Well With Wikidata? SSWS@ISWC 2015

References

[GQL22] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Oskar van Rest, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, Fred Zemke: Graph Pattern Matching in GQL and SQL/PGQ. SIGMOD

[SPARQL] Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. W3C Recommendation (2013)

[Cypher] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, Andrés Taylor: Cypher: An Evolving Query Language for Property Graphs. SIGMOD Conference 2018

[2RPQs] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Moshe Y. Vardi: Reasoning on regular path queries. SIGMOD Rec. 32(4) (2003)

[KRRV15] Egor V. Kostylev, Juan L. Reutter, Miguel Romero, Domagoj Vrgoč: SPARQL with Property Paths. ISWC 2015

[CM90] Mariano Consens, Alberto Mendelzon: GraphLog: a visual formalism for real life recursion. PODS 1990

References

[QDags] Diego Arroyuelo, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma: Optimal Joins Using Compressed Quadtrees. ACM TODS (2022)

[Ring] Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma, Adrián Soto: Worst-Case Optimal Graph Joins in Almost No Space. SIGMOD 2021

[LeapFrog] Todd L. Veldhuizen: Triejoin: A Simple, Worst-Case Optimal Join Algorithm. ICDT 2014

[Segoufin13]Luc Segoufin: Enumerating with constant delay the answers to a query, ICDT 2013

[REmatch] Cristian Riveros, Nicolás Van Sint Jan, Domagoj Vrgoč: REmatch: a novel regex engine for finding all matches. VLDB 2023

References

[RDF*] Olaf Hartig: Foundations of RDF⋆ and SPARQL⋆ (An Alternative Approach to Statement-Level Metadata in RDF). AMW 2017

[OneGraph] Ora Lassila, et. al.: The OneGraph vision: Challenges of breaking the graph model lock-in. Semantic Web 14(1): 125-134 (2023)

[AGM08] Albert Atserias, Martin Grohe, Dániel Marx: Size Bounds and Query Plans for Relational Joins. FOCS 2008: 739-748

[Ngo13] Hung Q. Ngo, Christopher Ré, Atri Rudra: Skew strikes back: new developments in the theory of join algorithms. SIGMOD Rec. 42(4): 5-16 (2013)

[Ngo18] Hung Q. Ngo: Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. PODS 2018: 111-124

Conclusions

Theoreticians got your back!

Two useful theoretical approaches

- Worst-case optimal (Leapfrog published in ICDT)
- Path queries (early PODS work)

An entire framework thought for practice

- Enumeration algorithms

Theoreticians can help practical work!

Try MillenniumDB

https://github.com/MillenniumDB/MillenniumDB

Thank you!