

# Topic 2: Complexity of Query Evaluation

## Unit 2: Beyond Conjunctive Queries

### Lecture 16

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp23)

<https://northeastern-datalab.github.io/cs7240/sp23/>

3/3/2023

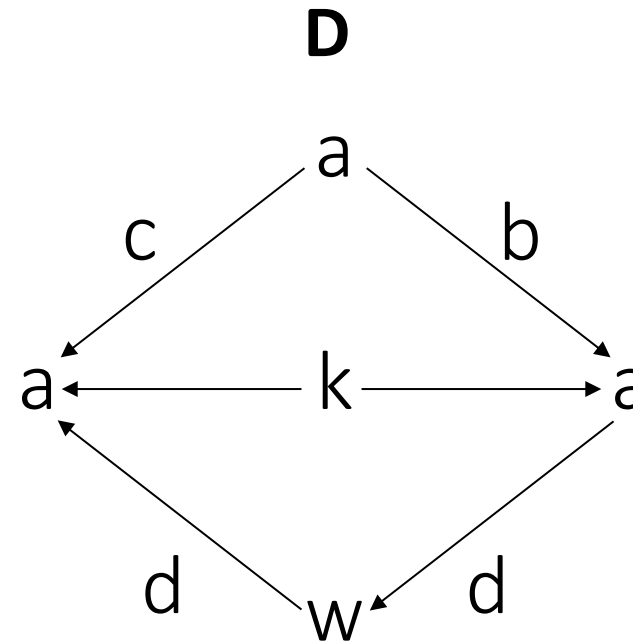
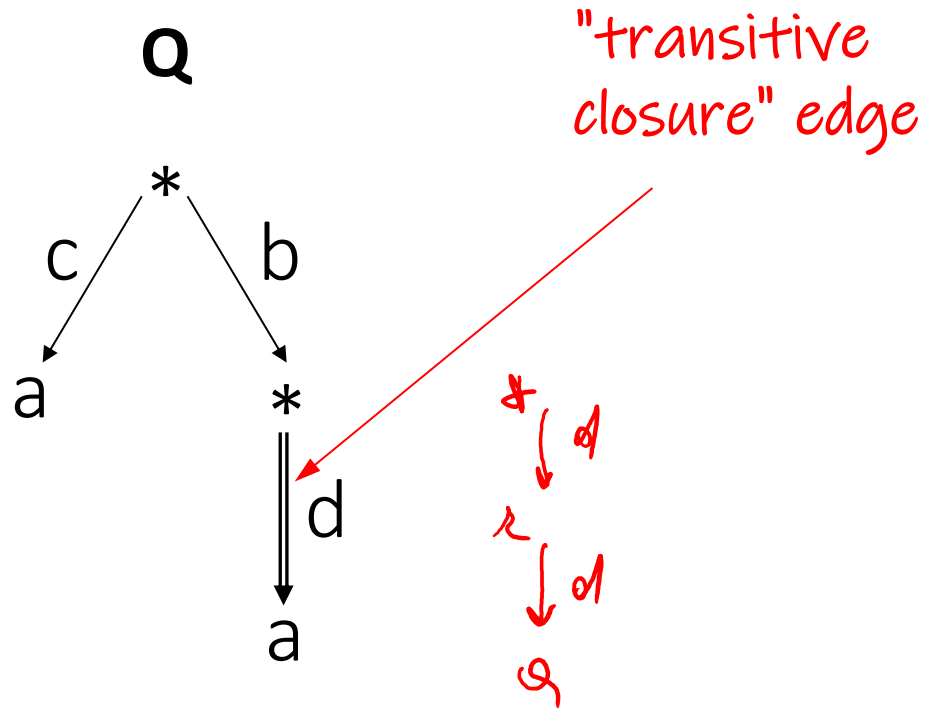
# Pre-class conversations

- Last class summary
- Project ideas
  
- Today:
  - Neha on the connection to CSPs (constraint satisfaction problems)
  - Beyond CQs: including open issues
  
  - tonight: more concrete projects
- Next week: spring break

# Outline: T2-1/2: Query Evaluation & Query Equivalence

- T2-1: Conjunctive Queries (CQs)
  - CQ equivalence and containment
  - Graph homomorphisms
  - Homomorphism beyond graphs
  - CQ containment
  - CQ minimization
- T2-2: Equivalence Beyond CQs
  - Union of CQs, and inequalities
  - Union of CQs equivalence under bag semantics
  - Tree pattern queries
  - Nested queries

# Tree pattern queries

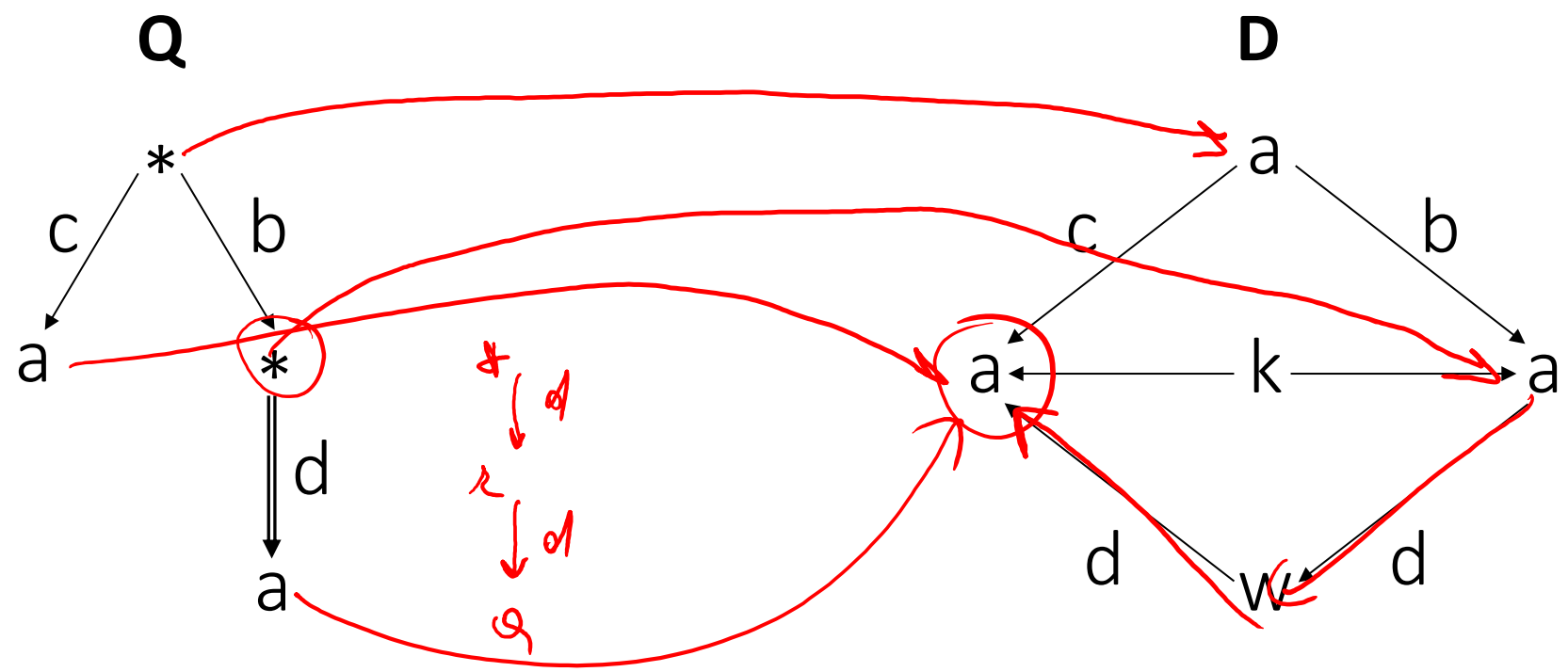


Does the query on the left have a match on in the data on the right (i.e. is there a homomorphism from left to right)?



Notice that "a", "b", "c" are labels (not node ids), thus like constants in a query, or like predicates (colored edges)

# Tree pattern queries



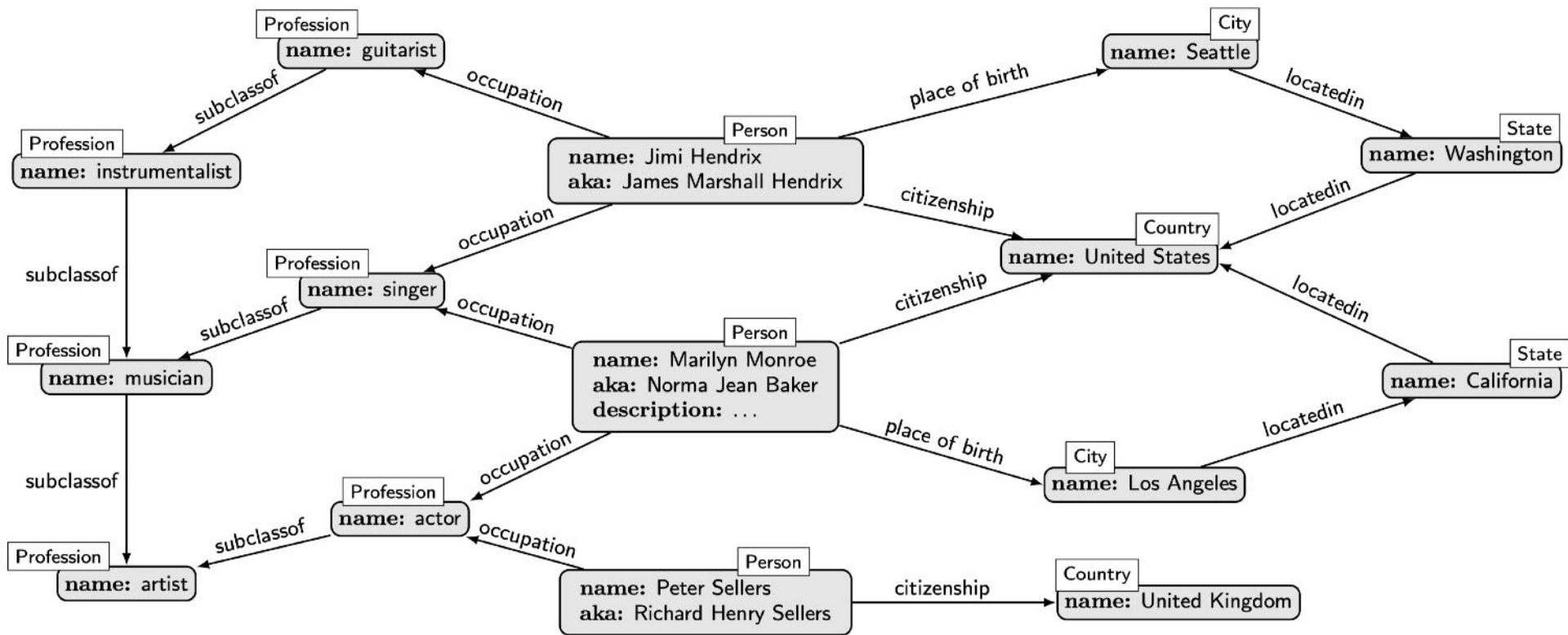


Figure 1: A graph database (as a *property graph*), inspired on a fragment of WikiData

?

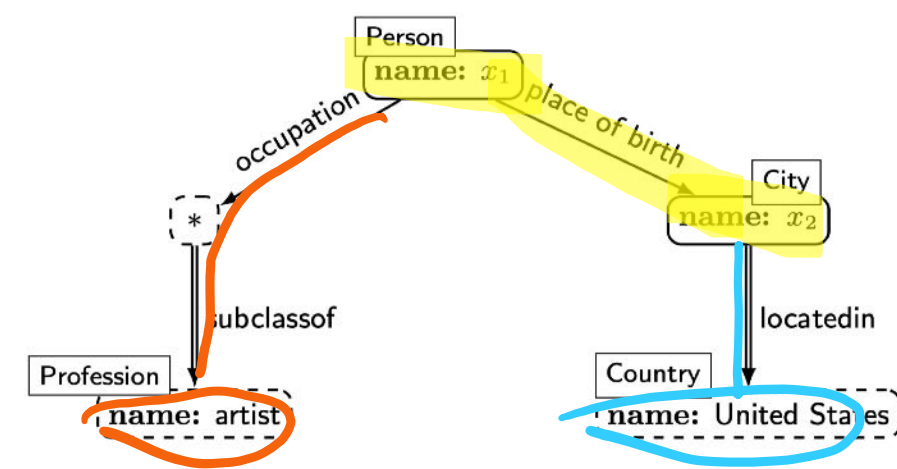


Figure 2: A tree pattern finding the artists who were born in the United States. The query returns the person names and the cities where they were born. (Fully circled nodes are return nodes.)

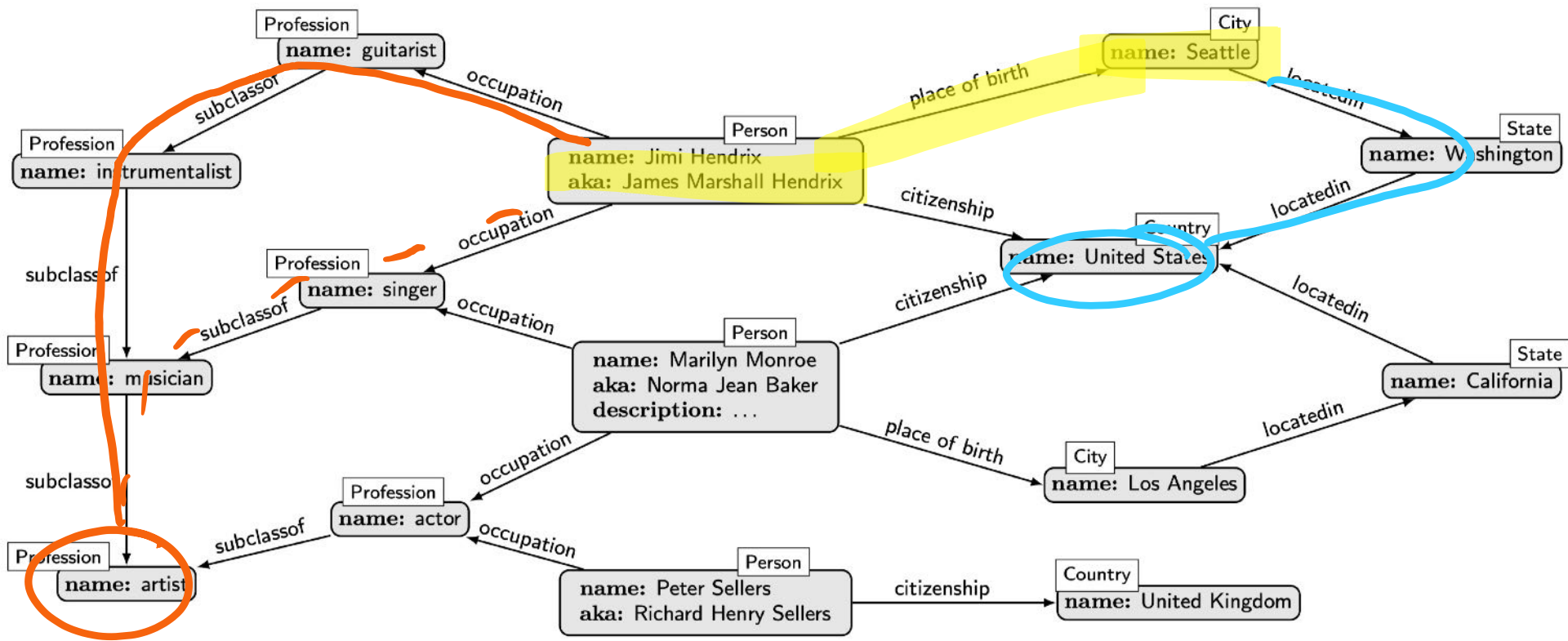


Figure 1: A graph database (as a *property graph*), inspired on a fragment of WikiData

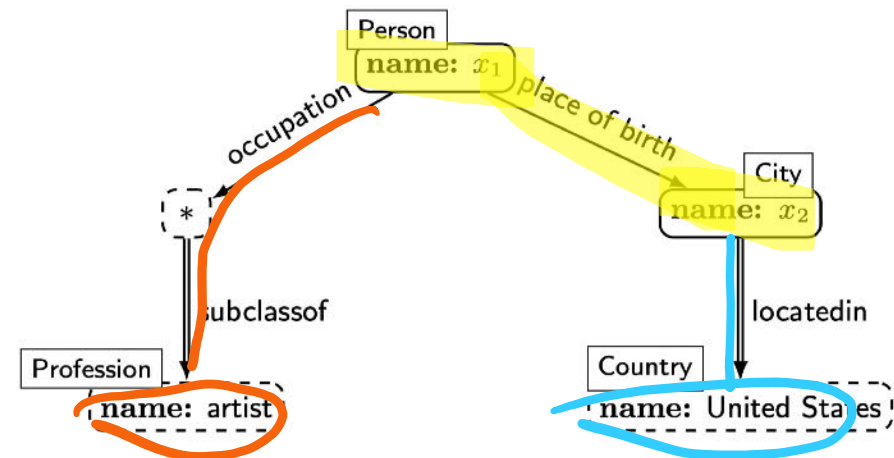


Figure 2: A tree pattern finding the artists who were born in the United States. The query returns the person names and the cities where they were born. (Fully circled nodes are return nodes.)

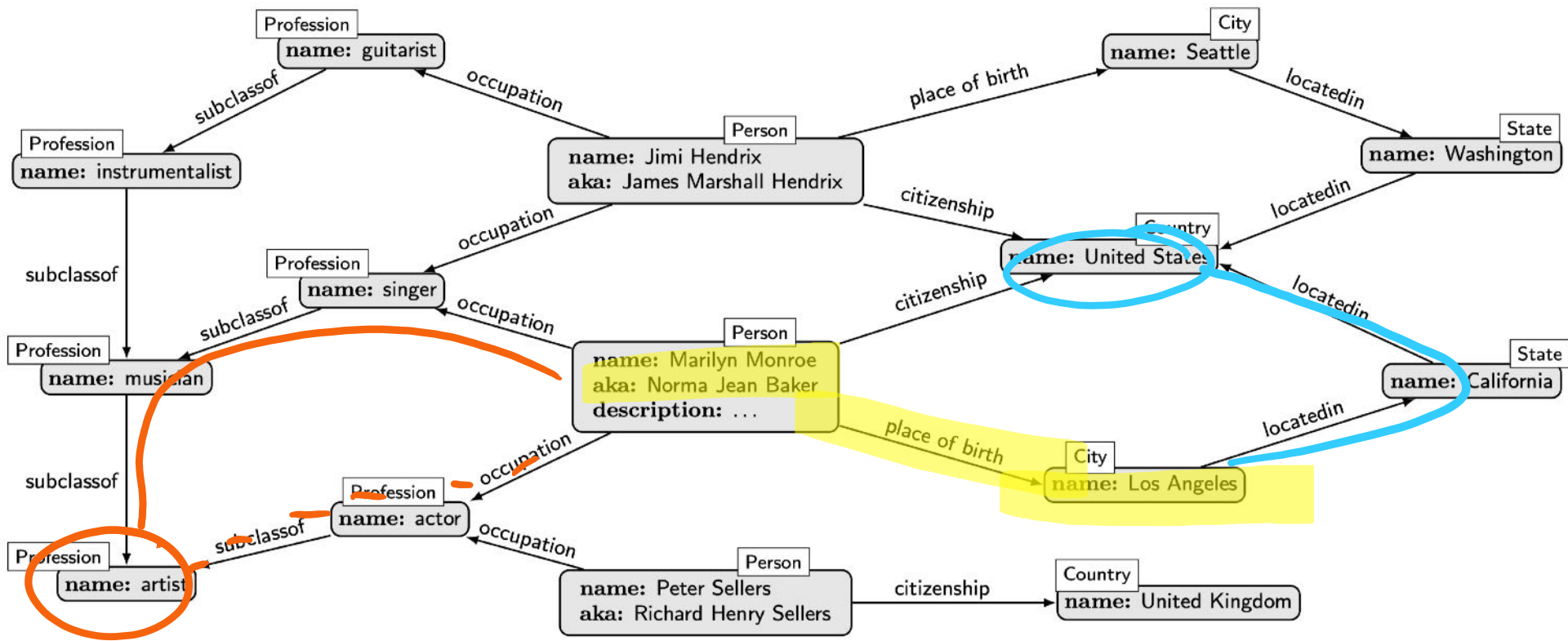


Figure 1: A graph database (as a *property graph*), inspired on a fragment of WikiData

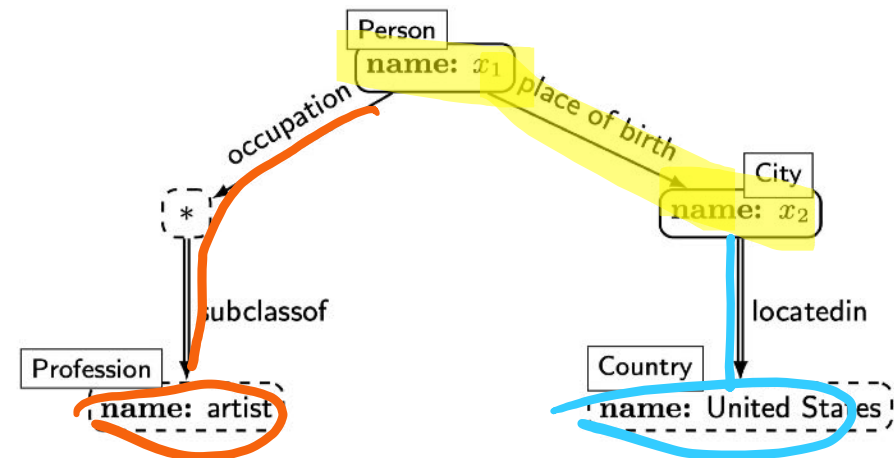
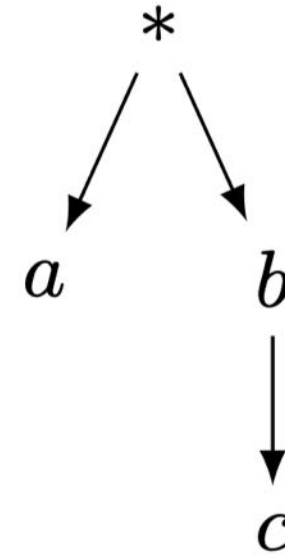
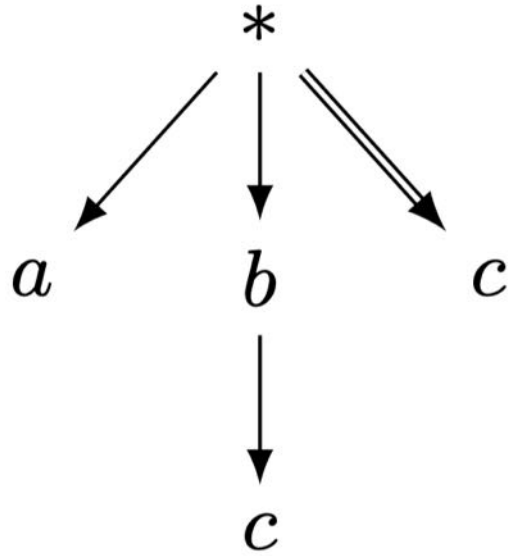


Figure 2: A tree pattern finding the artists who were born in the United States. The query returns the person names and the cities where they were born. (Fully circled nodes are return nodes.)



# Optimizing tree patterns



*How are those two tree patterns related to each other?*

?

# Optimizing tree patterns



## TREE PATTERN MINIMIZATION

Given: A tree pattern  $p$  and  $k \in \mathbb{N}$

Question: Is there a tree pattern  $q$ , equivalent to  $p$ , such that its size is at most  $k$ ?

# Minimality =? Nonredundancy

## 1.4 History of the Problem

Although the patterns we consider here have been widely studied [14, 24, 36, 15, 22, 1, 9, 4, 32], their minimization problem remained elusive for a long time. The most important previous work for their minimization was done by Kimelfeld and Sagiv [22] and by Flesca, Furfaro, and Masciari [14, 15].

The key challenge was understanding the relationship between *minimality* (M) and *nonredundancy* (NR). Here, a tree pattern is minimal if it has the smallest number of nodes among all equivalent tree patterns. It is nonredundant if none of its leaves (or branches<sup>2</sup>) can be deleted while remaining equivalent. The question was if minimality and nonredundancy are the same ([22, Section 7] and [15, p. 35]):

M  $\stackrel{?}{=}$  NR PROBLEM:

Is a tree pattern minimal  
if and only if it is nonredundant?

Notice that a part of the M  $\stackrel{?}{=}$  NR problem is easy to see: a minimal pattern is trivially also nonredundant (that is, M  $\subseteq$  NR). The opposite direction is much less clear.

If the problem would have a positive answer, it would mean that the simple algorithmic idea summarised in Algorithm 1 correctly minimizes tree patterns. Therefore, the M  $\stackrel{?}{=}$  NR problem is a natural question about the design of minimization algorithms for tree patterns.

---

### Algorithm 1 Computing a nonredundant subpattern

---

**Input:** A tree pattern  $p$

**Output:** A nonredundant tree pattern  $q$ , equivalent to  $p$

```
while a leaf of  $p$  can be removed  
    (remaining equivalent to  $p$ ) do  
    Remove the leaf  
end while  
return the resulting pattern
```

---

The  $M \stackrel{?}{=} \text{NR}$  problem is also a question about complexity. The main source of complexity of the nonredundancy algorithm lies in testing equivalence between a pattern  $p$  and a pattern  $p'$ , which is generally coNP-complete [24]. If  $M \stackrel{?}{=} \text{NR}$  has a positive answer, then TREE PATTERN MINIMIZATION would also be coNP-complete.

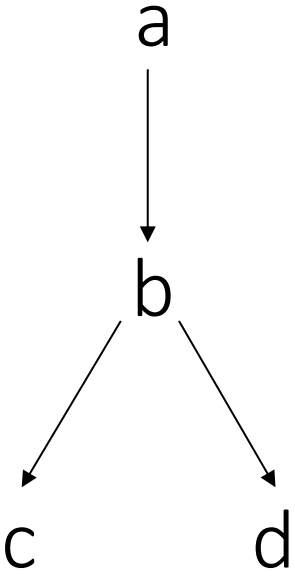
In fact, the problem was claimed to be coNP-complete in 2003 [14, Theorem 2], but the status of the minimization- and the  $M \stackrel{?}{=} \text{NR}$  problems were re-opened by Kimelfeld and Sagiv [22], who found errors in the proofs. Flesca et al.'s journal paper then proved that  $M = \text{NR}$  for a limited class of tree patterns, namely those where *every wildcard node has at most one child* [15]. Nevertheless, for tree patterns,

- (a) the status of the  $M \stackrel{?}{=} \text{NR}$  problem and
  - (b) the complexity of the minimization problem
- remained open.

Czerwinski, Martens, Niewerth, Parys [PODS 2016]

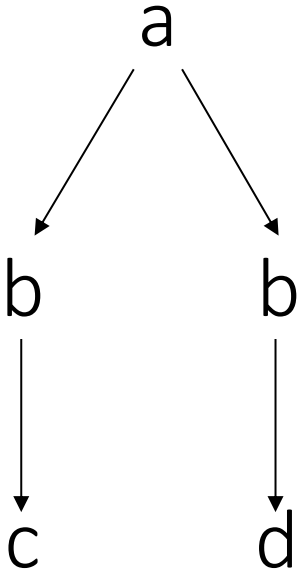
- (a) There exists a tree pattern that is nonredundant but not minimal. Therefore,  $M \neq \text{NR}$ .
- (b) TREE PATTERN MINIMIZATION is  $\Sigma_2^P$ -complete. This implies that even the main idea in Algorithm 1 cannot work unless  $\text{coNP} = \Sigma_2^P$ .

# Tree pattern containment

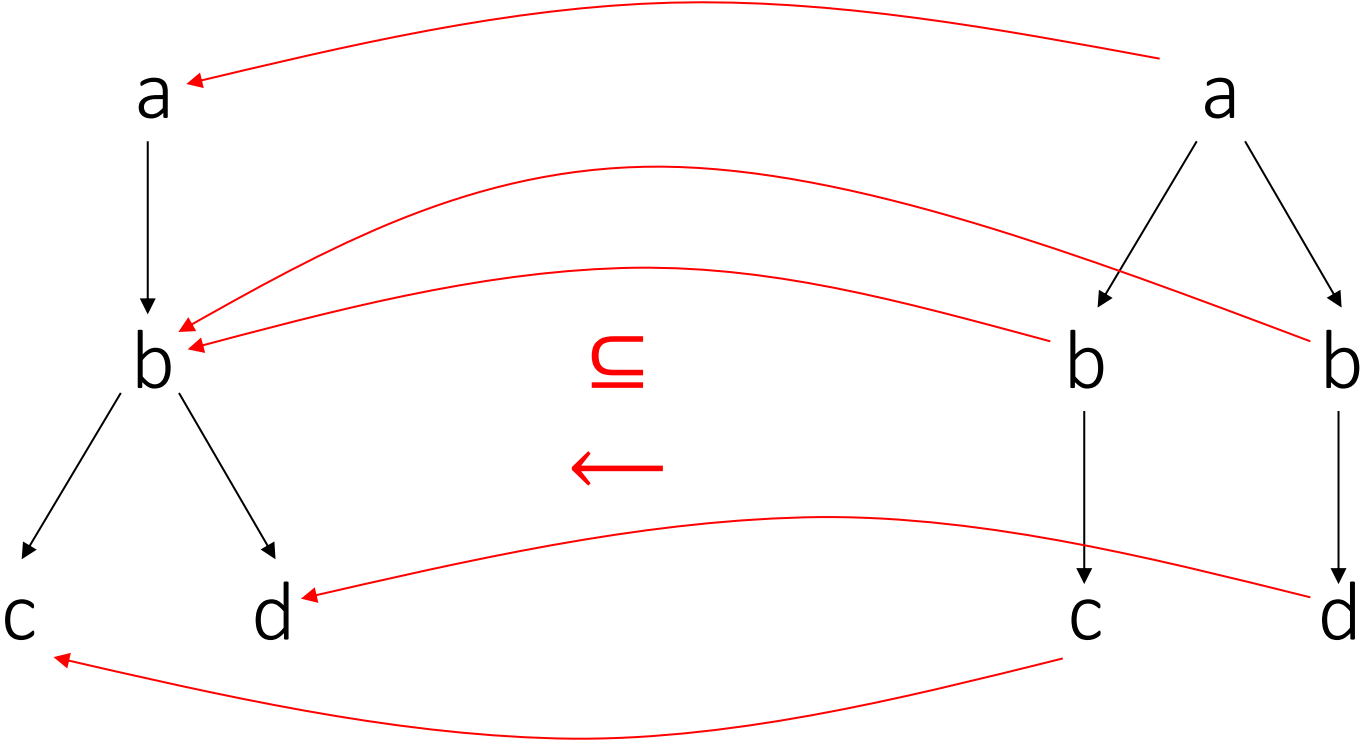


$\supseteq$   
or  
 $\supseteq$

?



# Tree pattern containment



but  $\neq$ !

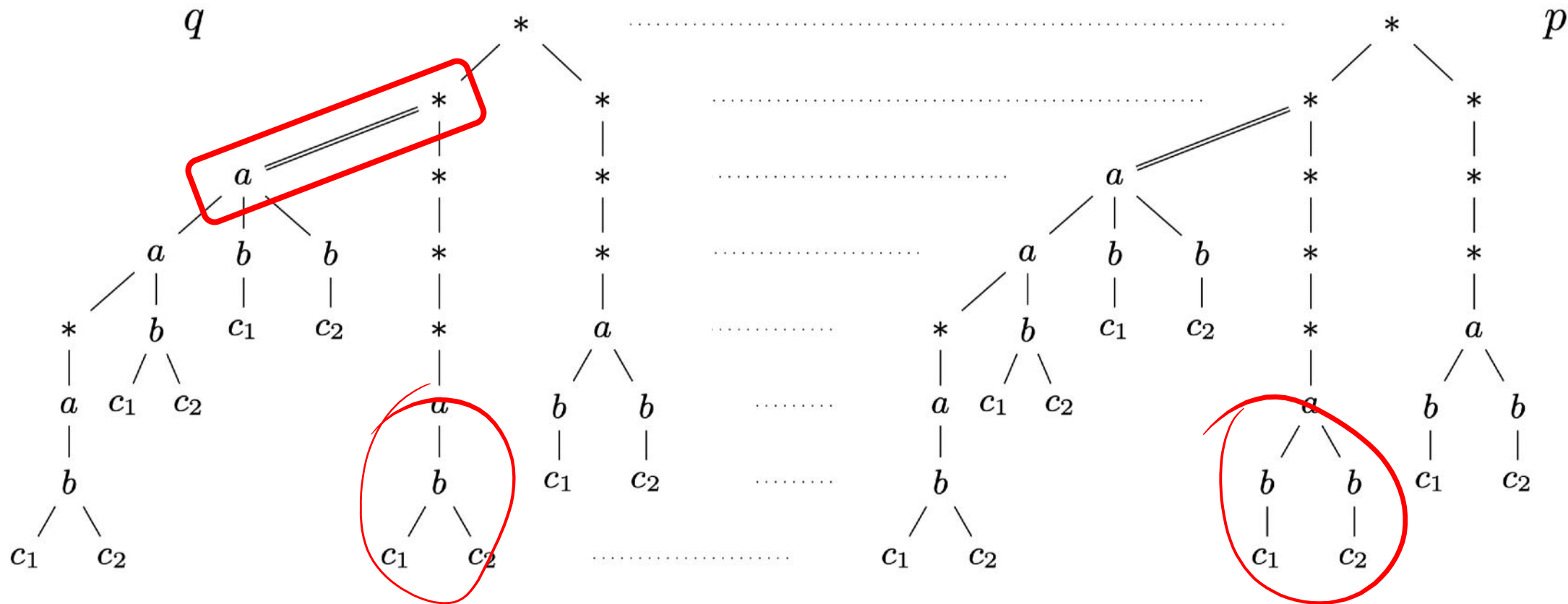
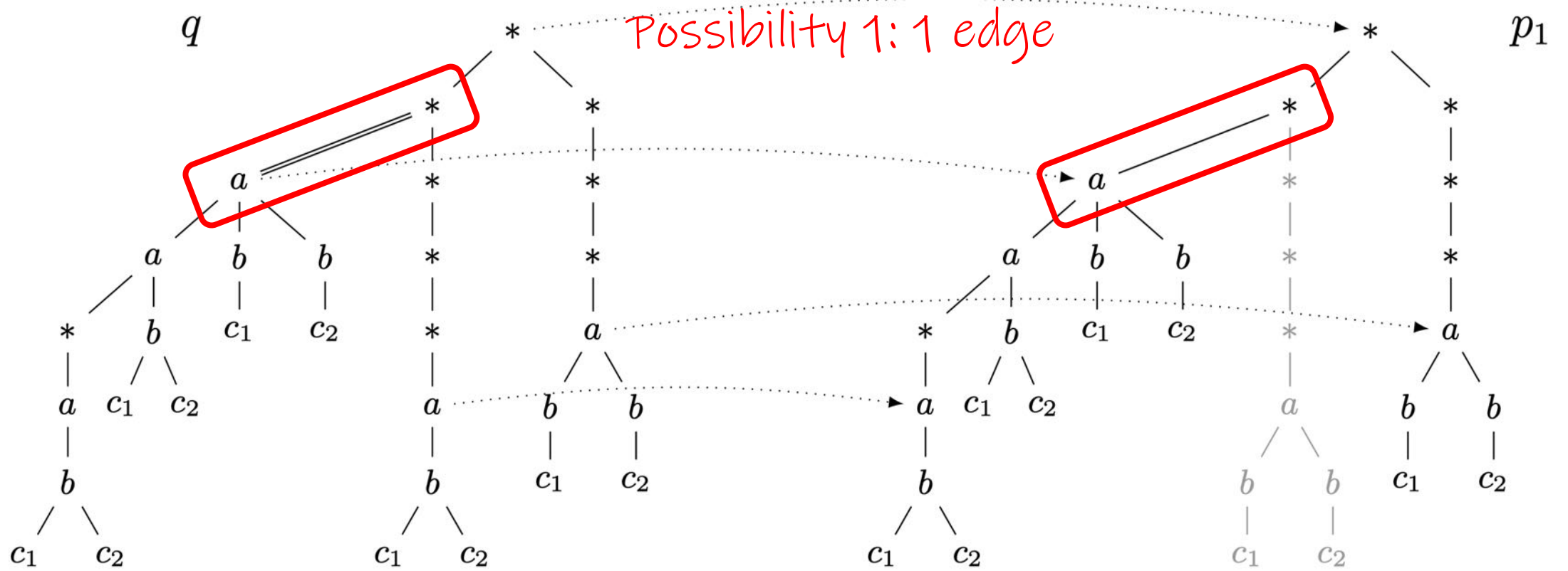


Figure 7: A non-redundant tree pattern  $p$  (right) and an equivalent tree pattern  $q$  that is smaller (left)

$q \subseteq p$  follows from argument on previous page.

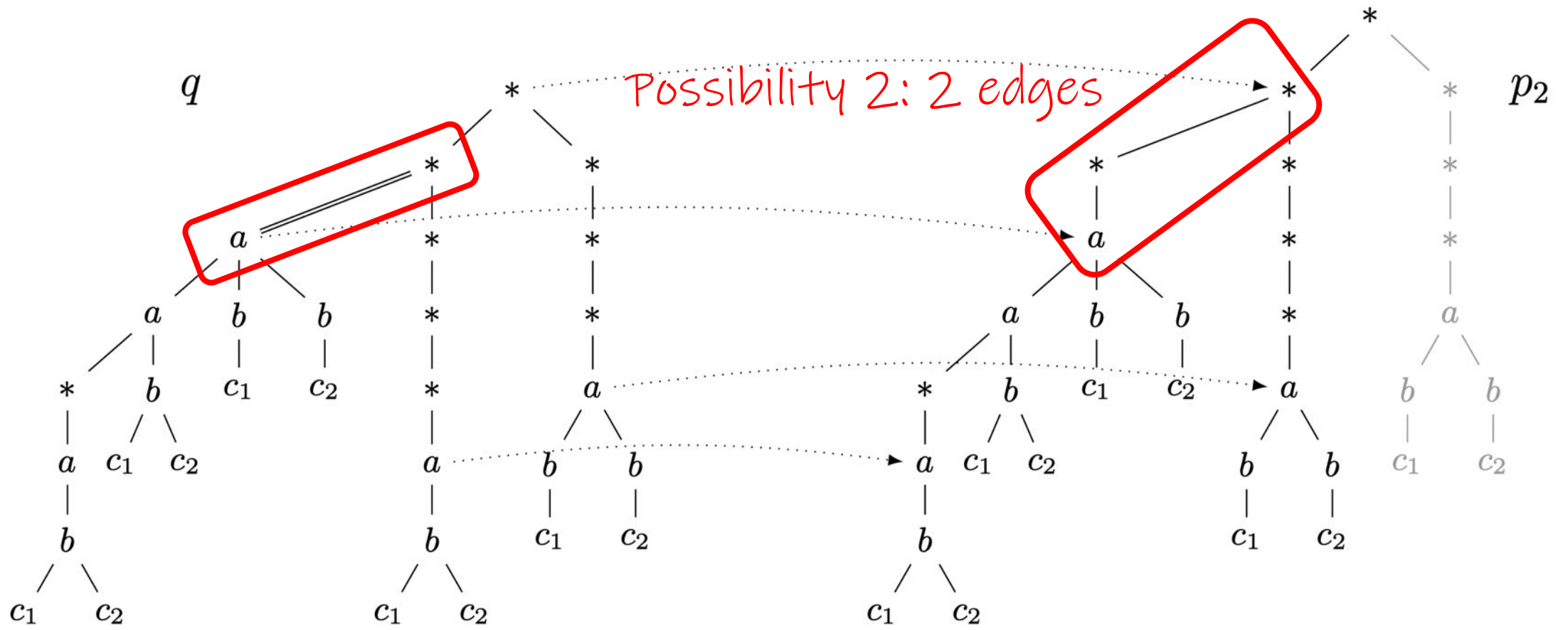
To be shown  $q \supseteq p$ , then equivalent. Idea: whenever  $p$  matches, then also  $q$ .

Idea:  $a = *$  can be matched in 3 ways in a graph

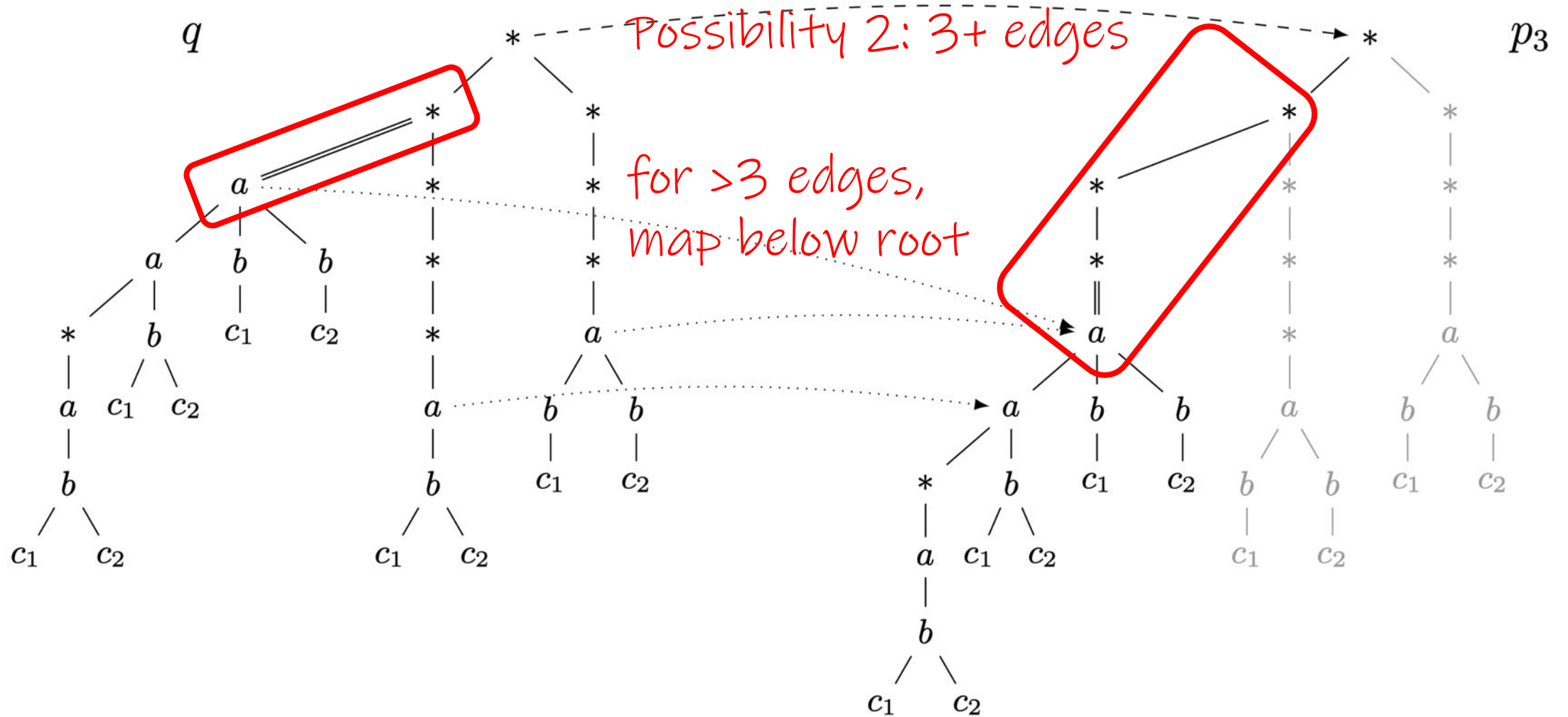


(a) How  $q$  can be matched if  $p_1$  can be matched





(b) How  $q$  can be matched if  $p_2$  can be matched



(c) How  $q$  can be matched if  $p_3$  can be matched

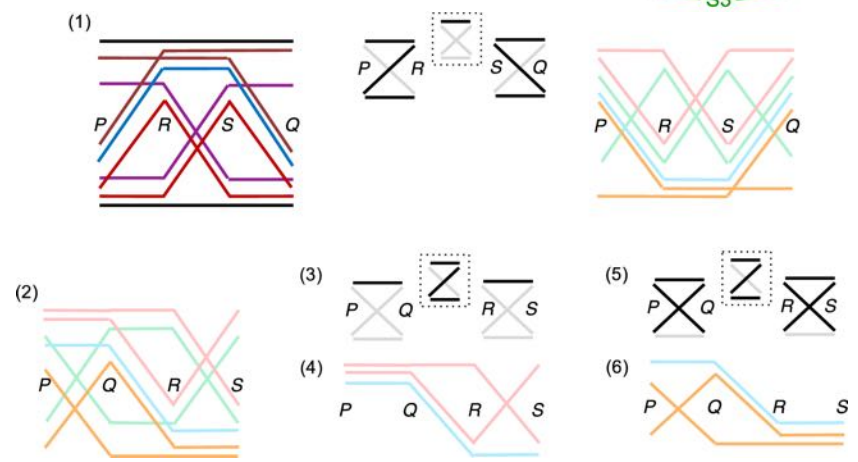
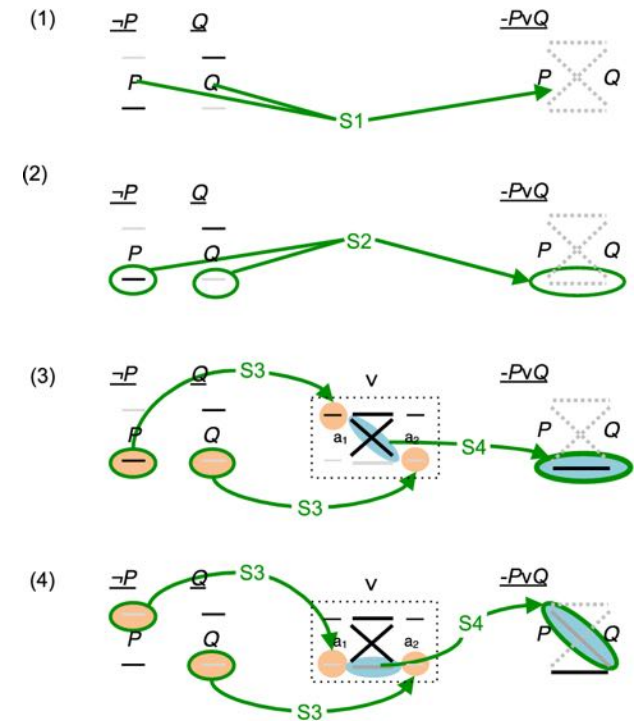
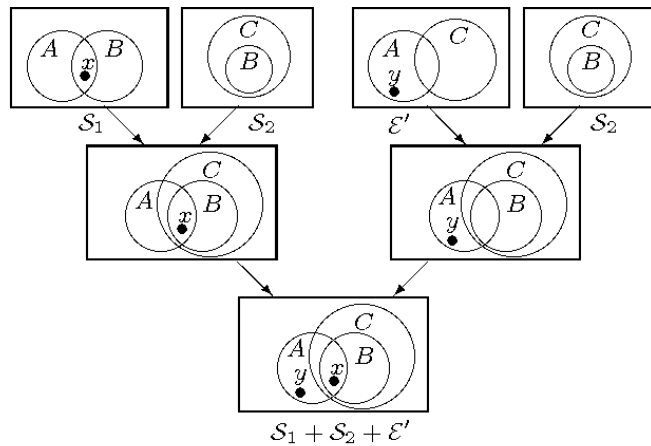
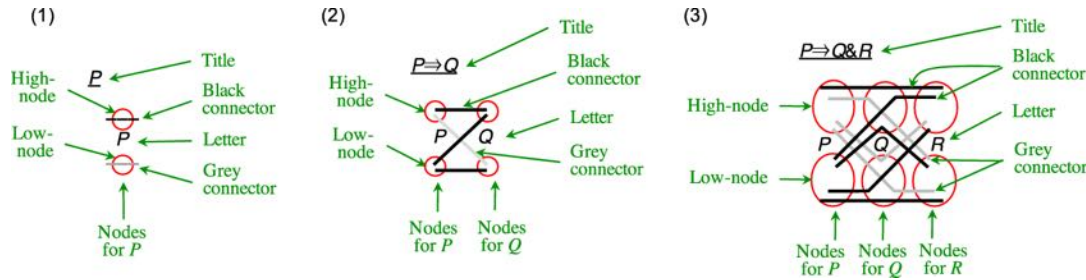
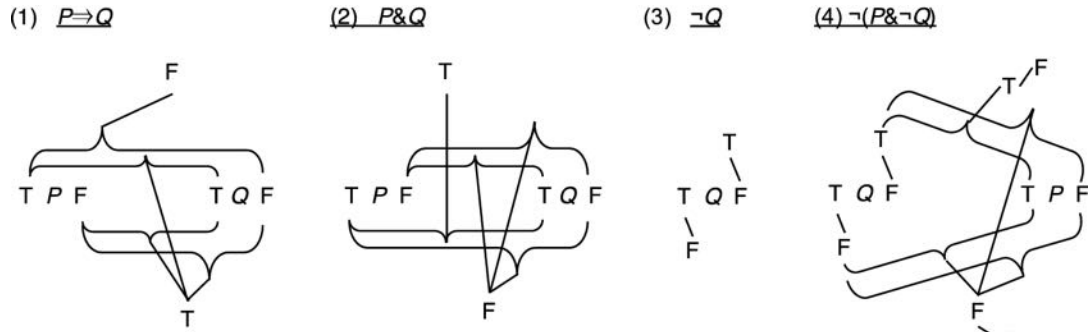
# Outline: T2-1/2: Query Evaluation & Query Equivalence

- T2-1: Conjunctive Queries (CQs)
  - CQ equivalence and containment
  - Graph homomorphisms
  - Homomorphism beyond graphs
  - CQ containment
  - CQ minimization
- T2-2: Equivalence Beyond CQs
  - Union of CQs, and inequalities
  - Union of CQs equivalence under bag semantics
  - Tree pattern queries
  - Nested queries

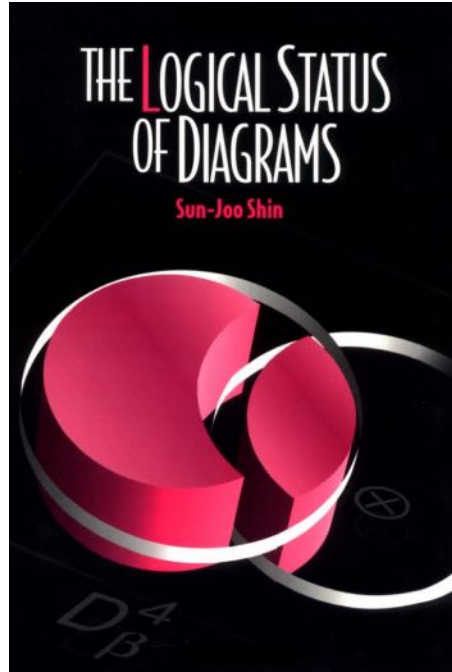
# Equivalence of nested queries

- **Query equivalence** is one of the foundational questions in database theory (and practice?)
  - touches on logics and decidability
  - what modifications allow tractability
- Lots of work (and open questions) on query equivalence
  - But not so much on **nested queries**!
- Related to QueryVis project (<http://queryvis.com/>) and two foundational questions on visual formalism:
  1. When can visual formalism *unambiguously* express logical statements?
  2. When can equivalent logical statements be transformed to each other by a sequence of visual transformations? (*Query equivalence*)

# Diagrammatic reasoning systems and their expressiveness



# Diagrammatic reasoning systems and their expressiveness



**Diagrams** are widely used in reasoning about problems in physics, mathematics, and logic, but have traditionally been considered to be only heuristic tools and not valid elements of mathematical proofs. This book challenges this prejudice against visualization in the history of logic and mathematics and provides a formal foundation for work on natural reasoning in a visual mode.

The author presents Venn diagrams as a formal system of representation equipped with its own syntax and semantics and specifies rules of transformation that make this system sound and complete. The system is then extended to the equivalent of a first-order monadic language. The soundness of these diagrammatic systems refutes the contention that graphical representation is misleading in reasoning. The validity of the transformation rules ensures that the correct application of the rules will not lead to fallacies. The book concludes with a discussion of some fundamental differences between graphical systems and linguistic systems.

This groundbreaking work will have important influence on research in logic, philosophy, and knowledge representation.

objects. **Conjunctive information** is more naturally represented by diagrams than by linguistic formulæ. For example, a single Venn diagram can

Still, not all relations can be viewed as membership or inclusion. Shin has been careful throughout her book to **restrict herself to monadic systems**. Relations per se (polyadic predicates) are not considered. And while it may be true that the formation of a system (such as Venn-II) that is provably both sound and complete would help mitigate the prejudice

perception. In her discussion of perception she shows that **disjunctive information is not representable in any system**. In doing so she relies on

# QueryVis

- Motivation: Can we create an automatic diagramming system that:
  - unambiguously visualizes the logical intent of a relational query (thus no two different queries lead to an “identical” visualization; with “identical” to be formalized correctly)
  - for some important subset of nested queries (later extensions from SQL)
  - with visual diagrams that allow us to reason about **logical SQL design patterns**
- Related:
  - Lot’s of interest on conjunctive queries equivalence. Now: For what fragment of nested queries is equivalence decidable (under set semantics)?
- Suggestion:
  - nested queries, with inequalities, without any disjunctions
  - Strict superset of conjunctive queries

# Logical SQL Patterns

**Logical patterns** are the building blocks of most SQL queries.

Patterns are very hard to extract from the SQL text.

A pattern can appear across different database schemas.

Think of queries like:

- Find sailors who reserved all red boats
- Find students who took all art classes
- Find actors who played in all movies by Hitchcock



# What does this query return ?

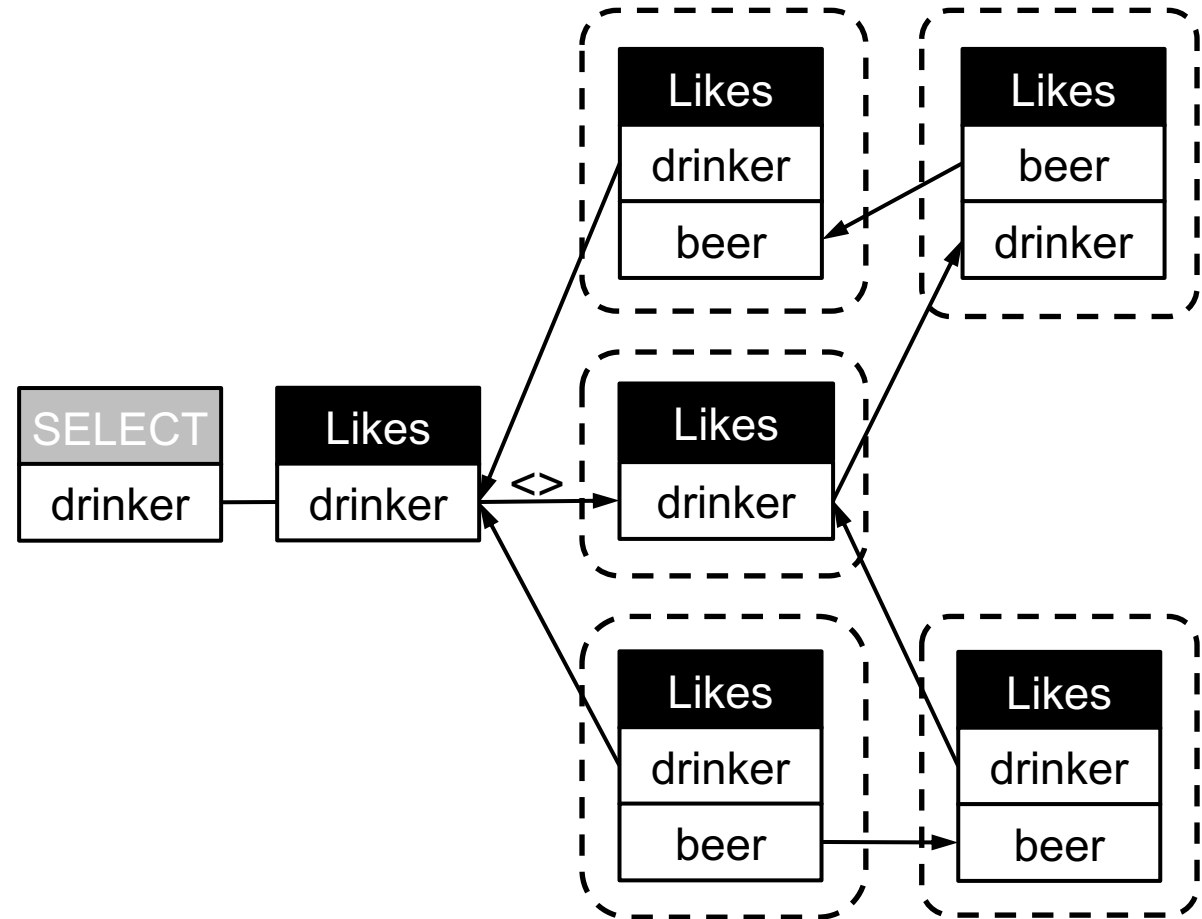
Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
   FROM Likes L2
   WHERE L1.drinker <> L2.drinker
   AND not exists
     (SELECT *
      FROM Likes L3
      WHERE L3.drinker = L2.drinker
      AND not exists
        (SELECT *
         FROM Likes L4
         WHERE L4.drinker = L1.drinker
         AND L4.beer = L3.beer)))
AND not exists
  (SELECT *
   FROM Likes L5
   WHERE L5.drinker = L1.drinker
   AND not exists
     (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer = L5.beer)))
```

# What does this query return

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
   FROM Likes L2
   WHERE L1.drinker <> L2.drinker
   AND not exists
     (SELECT *
      FROM Likes L3
      WHERE L3.drinker = L2.drinker
      AND not exists
        (SELECT *
         FROM Likes L4
         WHERE L4.drinker = L1.drinker
         AND L4.beer = L3.beer)))
AND not exists
  (SELECT *
   FROM Likes L5
   WHERE L5.drinker = L1.drinker
   AND not exists
     (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer = L5.beer)))
```

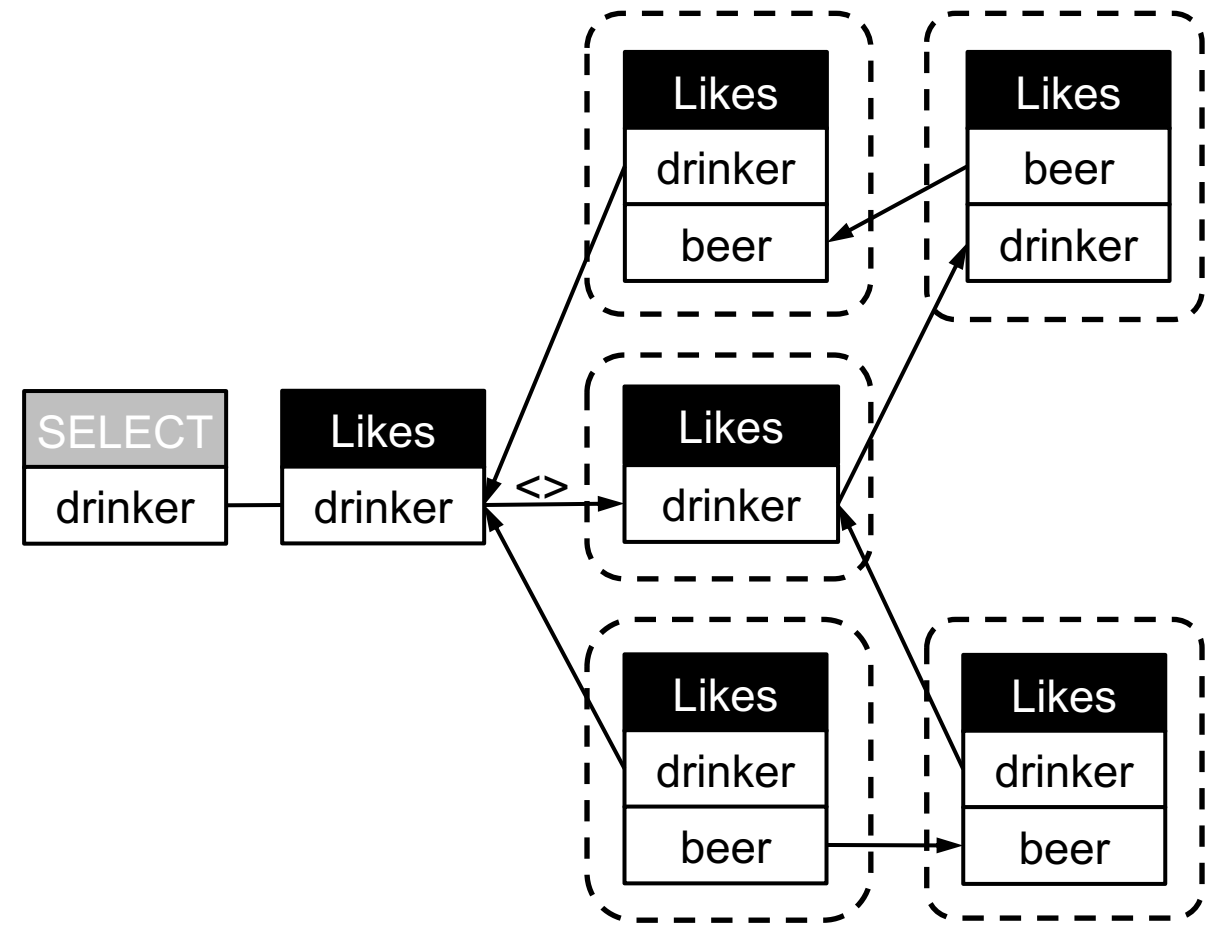


QueryVis scoping

# Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
   FROM Likes L2
   WHERE L1.drinker <> L2.drinker
   AND not exists
     (SELECT *
      FROM Likes L3
      WHERE L3.drinker = L2.drinker
      AND not exists
        (SELECT *
         FROM Likes L4
         WHERE L4.drinker = L1.drinker
         AND L4.beer = L3.beer)))
AND not exists
  (SELECT *
   FROM Likes L5
   WHERE L5.drinker = L1.drinker
   AND not exists
     (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer = L5.beer)))
```

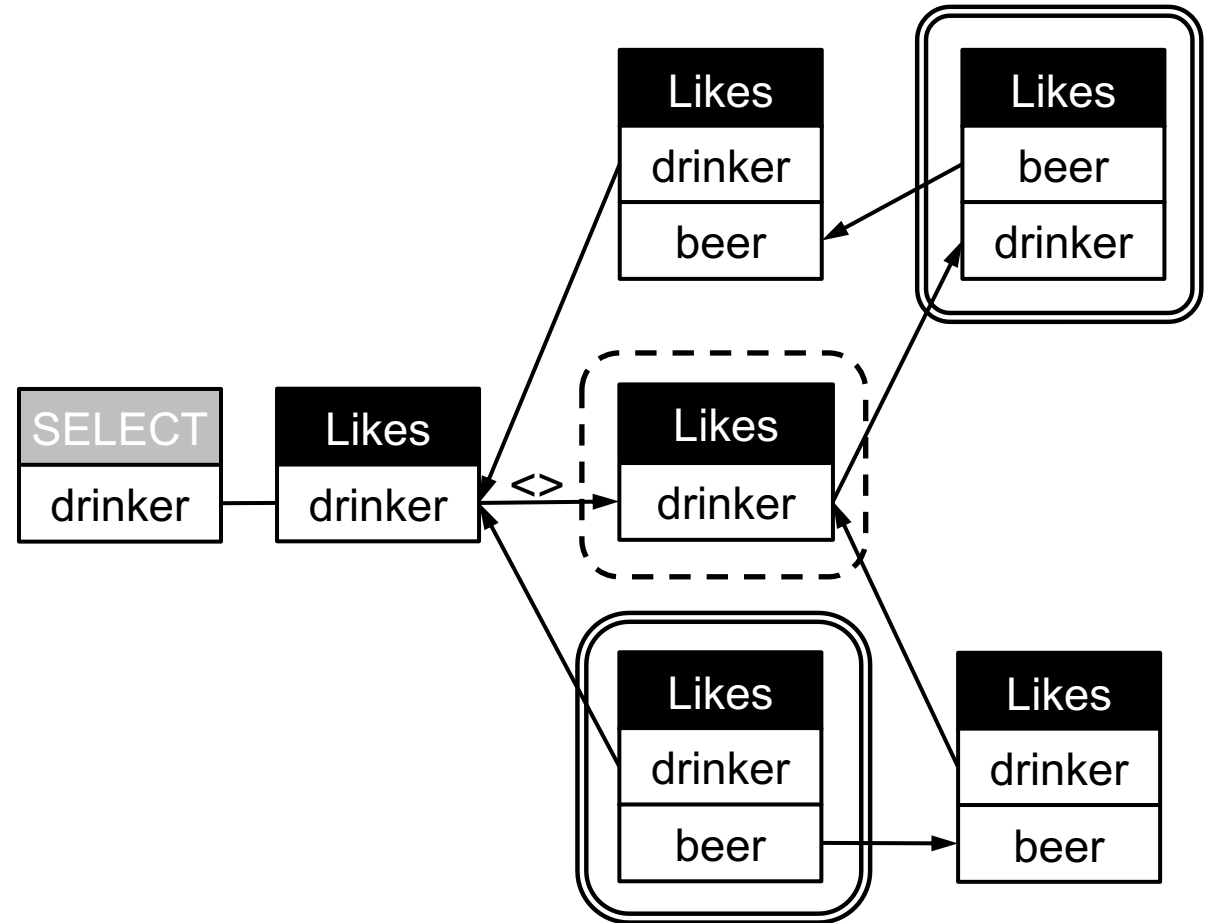


QueryVis scoping

# Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
   FROM Likes L2
   WHERE L1.drinker <> L2.drinker
   AND not exists
     (SELECT *
      FROM Likes L3
      WHERE L3.drinker = L2.drinker
      AND not exists
        (SELECT *
         FROM Likes L4
         WHERE L4.drinker = L1.drinker
         AND L4.beer = L3.beer)))
AND not exists
  (SELECT *
   FROM Likes L5
   WHERE L5.drinker = L1.drinker
   AND not exists
     (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer = L5.beer)))
```

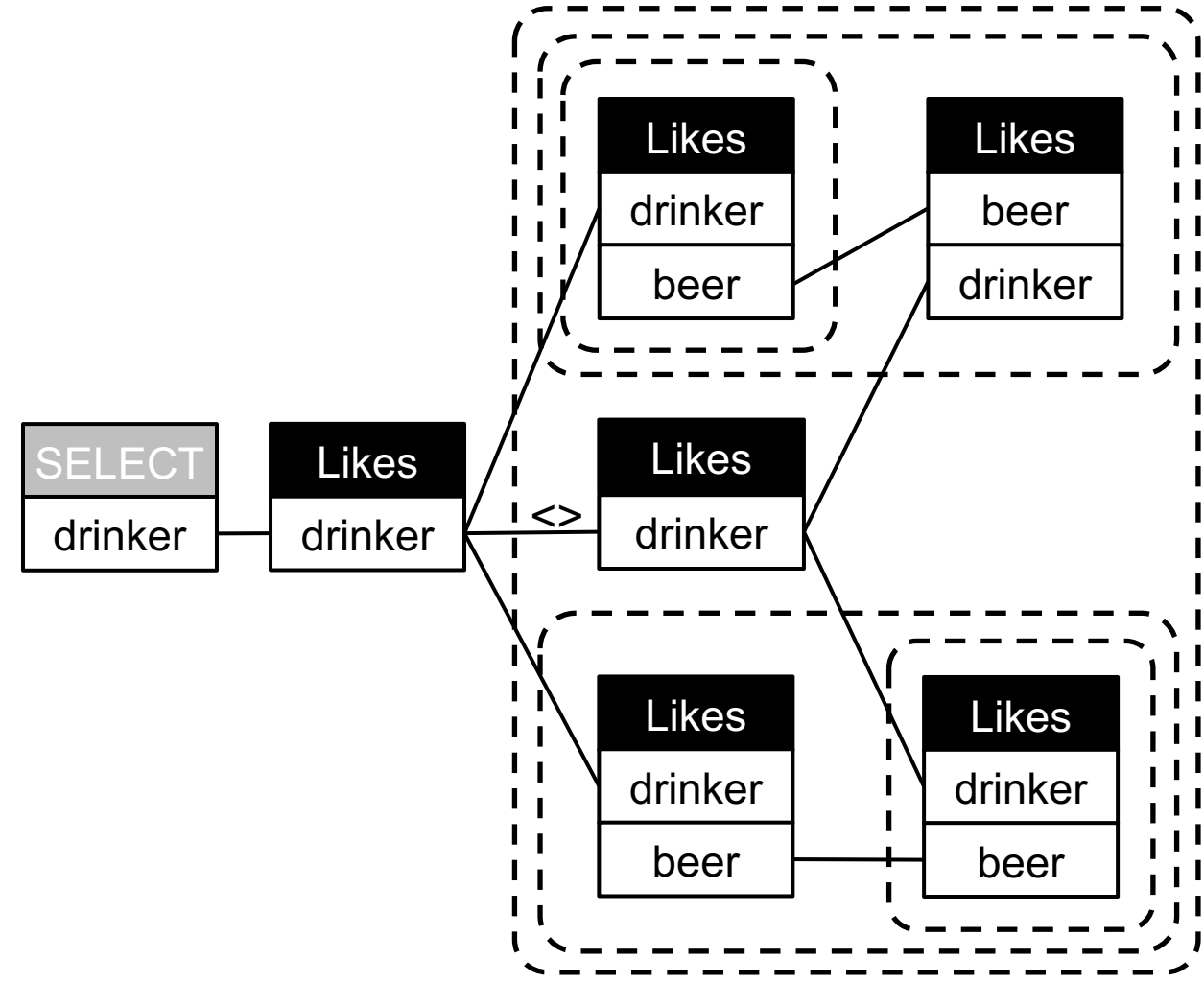


QueryVis scoping

# Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
   FROM Likes L2
   WHERE L1.drinker <> L2.drinker
   AND not exists
     (SELECT *
      FROM Likes L3
      WHERE L3.drinker = L2.drinker
      AND not exists
        (SELECT *
         FROM Likes L4
         WHERE L4.drinker = L1.drinker
         AND L4.beer = L3.beer)))
AND not exists
  (SELECT *
   FROM Likes L5
   WHERE L5.drinker = L1.drinker
   AND not exists
     (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer = L5.beer)))
```



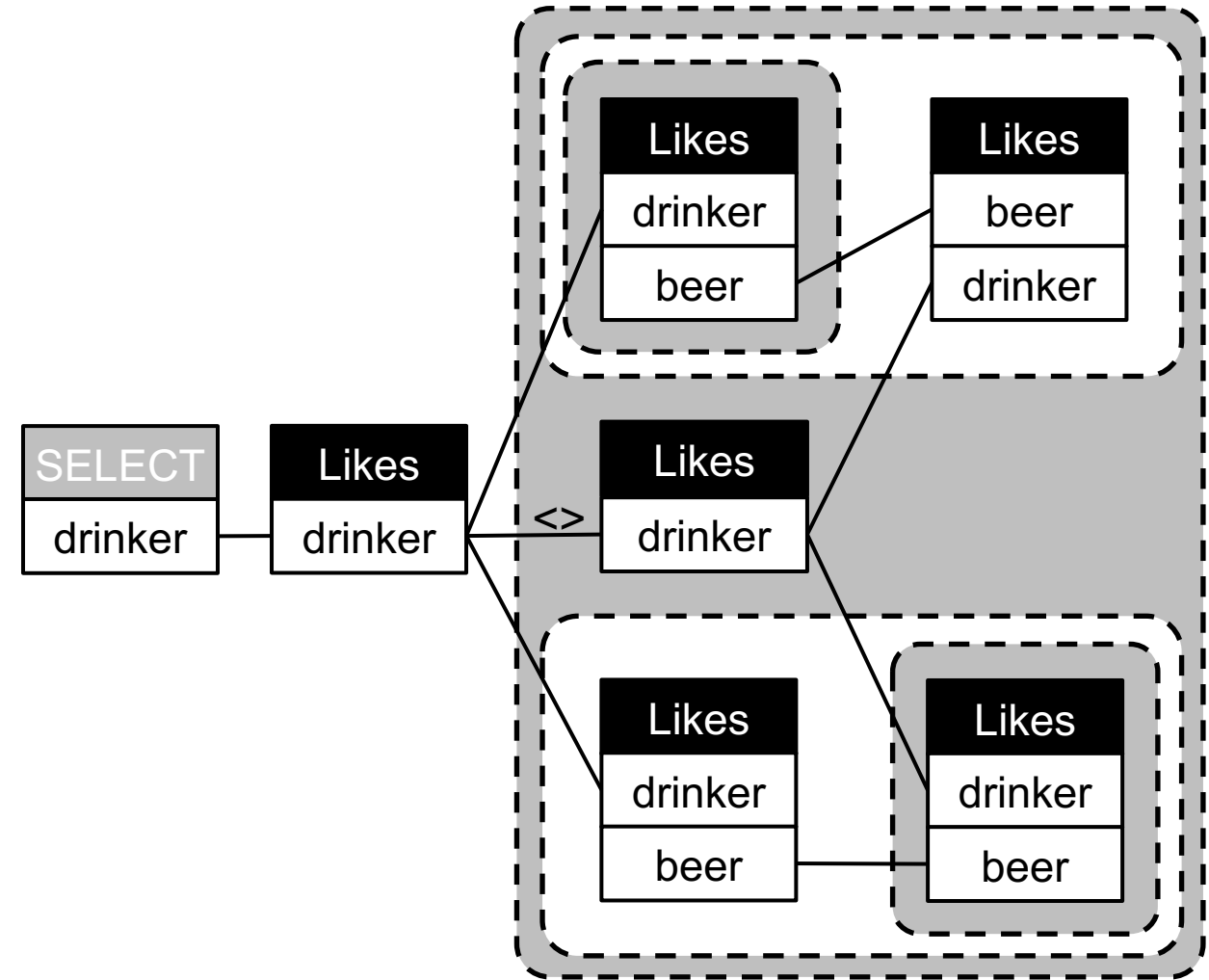
QueryVis scoping

Relational Diagrams scoping (<https://relationaldiagrams.com>)

# Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
   FROM Likes L2
   WHERE L1.drinker <> L2.drinker
   AND not exists
     (SELECT *
      FROM Likes L3
      WHERE L3.drinker = L2.drinker
      AND not exists
        (SELECT *
         FROM Likes L4
         WHERE L4.drinker = L1.drinker
         AND L4.beer = L3.beer)))
AND not exists
  (SELECT *
   FROM Likes L5
   WHERE L5.drinker = L1.drinker
   AND not exists
     (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer = L5.beer)))
```



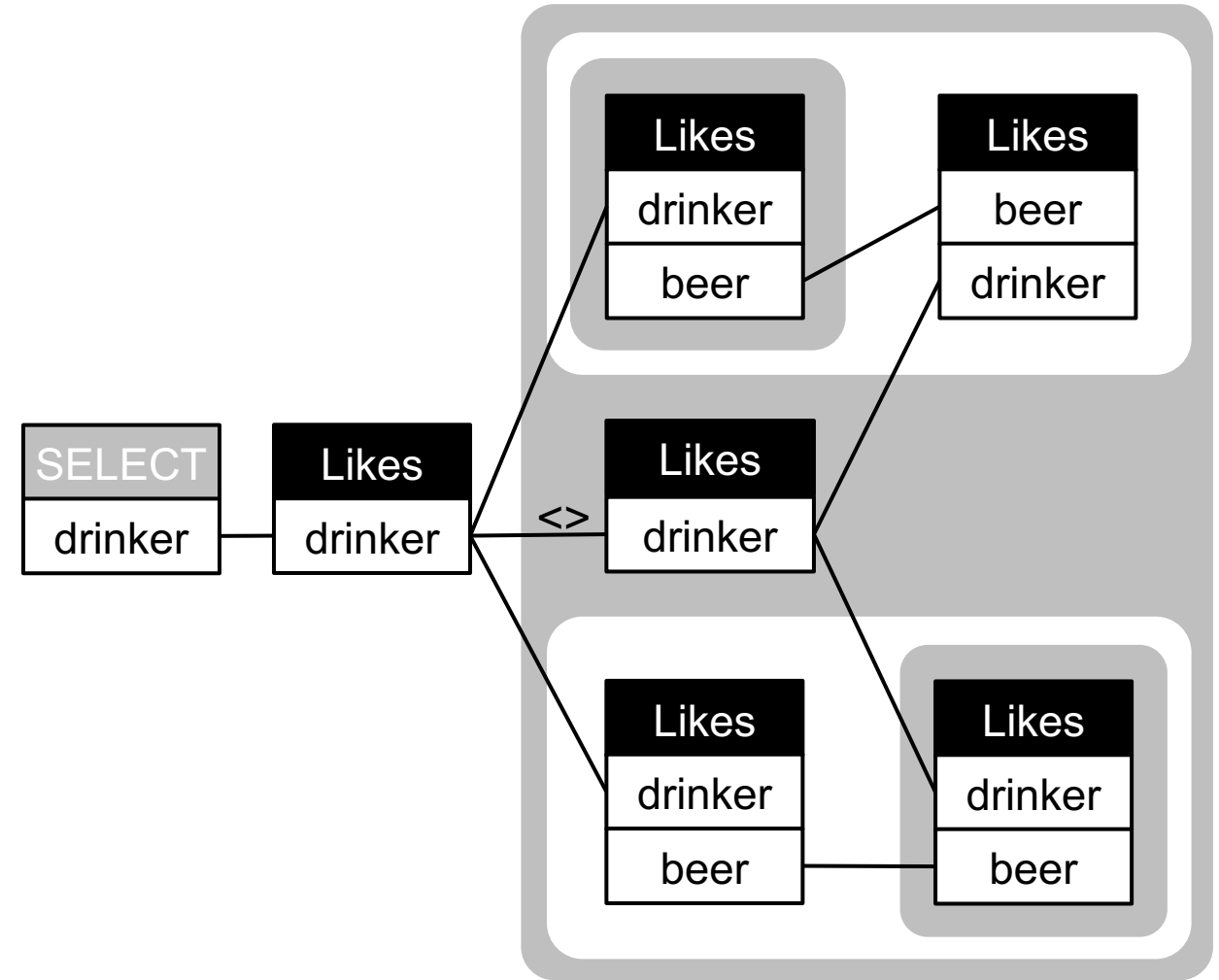
QueryVis scoping

Relational Diagrams scoping (<https://relationaldiagrams.com>)

# Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
   FROM Likes L2
   WHERE L1.drinker <> L2.drinker
   AND not exists
     (SELECT *
      FROM Likes L3
      WHERE L3.drinker = L2.drinker
      AND not exists
        (SELECT *
         FROM Likes L4
         WHERE L4.drinker = L1.drinker
         AND L4.beer = L3.beer)))
AND not exists
  (SELECT *
   FROM Likes L5
   WHERE L5.drinker = L1.drinker
   AND not exists
     (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer = L5.beer)))
```



QueryVis scoping

Relational Diagrams scoping (<https://relationaldiagrams.com>)

<https://demo.queryvis.com>

# QueryViz

Input: Schema

Input Query

Output: Visualization

**Your Input**

Specify or choose a pre-defined schema help

Employee and Department

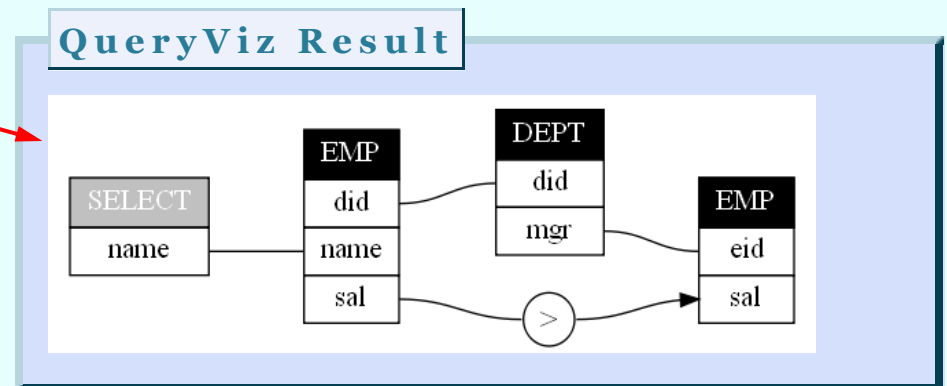
```
EMP(eid,name,sal,did)
DEPT(did,dname,mgr)
```

Specify or choose an SQL Query help

Query 8

```
SELECT e1.name
FROM EMP e1, EMP e2, DEPT d
WHERE e1.did = d.did
AND d.mgr = e2.eid
AND e1.sal > e2.sal
```

Submit



Danaparamita, G. [EDBT'11]

<https://queryvis.com/>

<http://www.youtube.com/watch?v=kVFnQRGAQIs>

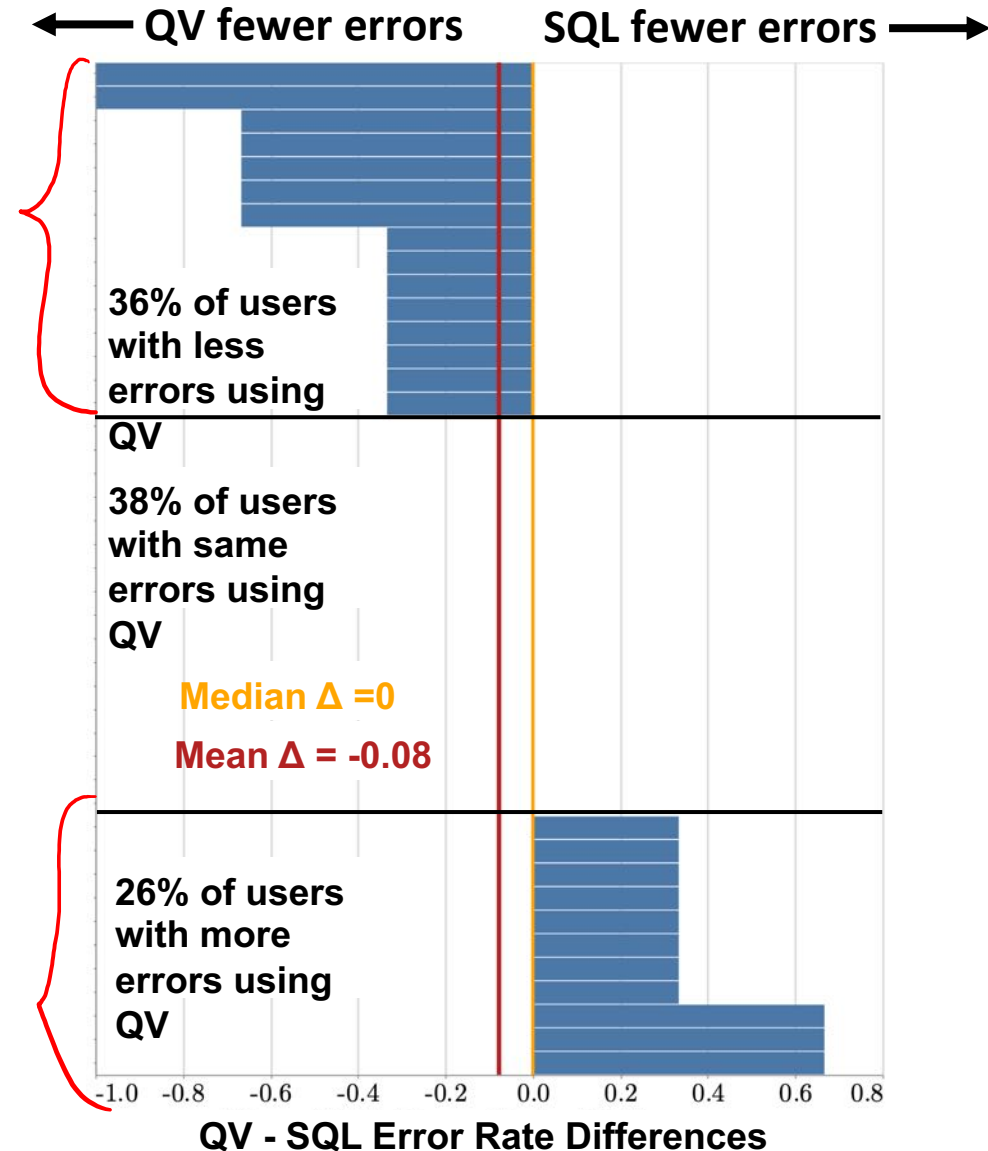
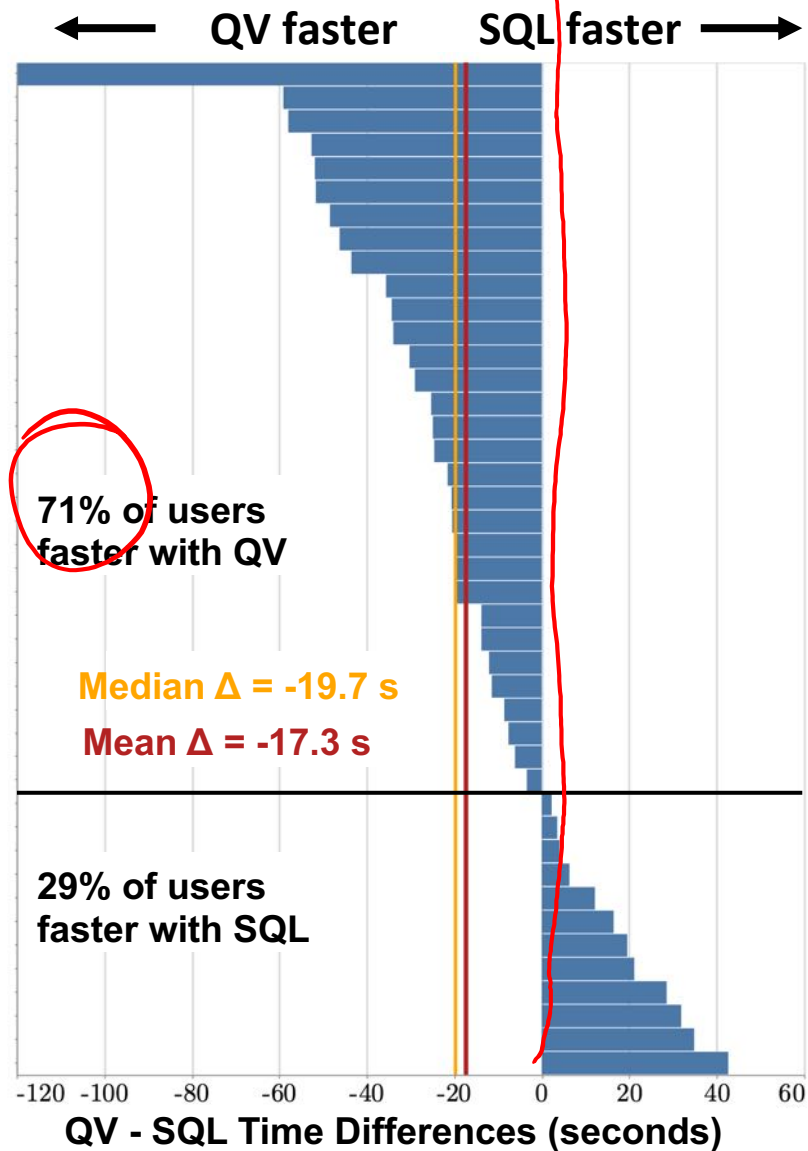
Source: Danaparamita, Gatterbauer: QueryViz: Helping users understand SQL queries and their patterns. EDBT 2011. <https://doi.org/10.14778/3402755.3402805>

Wolfgang Gatterbauer. Principles of scalable data management: <https://northeastern-datalab.github.io/cs/z4u/>



# Amazon Turk user study with SQL users

Each bar below corresponds to one participant (42 bars/participants in total)



Northeastern University

# DATA Lab @ Northeastern

Scalable Management and Analysis of Big Data

- Home
- People
- Research Opportunities
- Recent Publications
- Activities
- YouTube Channel**

## DATA LAB @ NORTHEASTERN

The Data Lab @ Northeastern University is one of the leading research groups in data management and data systems. Our work spans the breadth of data management, from the foundations of data integration and curation, to large-scale and parallel data-centric computing. Recent research projects include query visualization, data provenance, data discovery, data lake management, and scalable approaches to perform inference over uncertain

<https://queryvis.com>

**THE STORY OF QUERYVIS, NOT JUST ANOTHER VISUAL PROGRAMMING LANGUAGE**

TUE 06.30.20 / YSABELLE KEMPE

<https://www.khoury.northeastern.edu/the-story-of-queryvis-not-just-another-visual-programming-language/>

Unique set query: "Find drinkers that like a unique set of beers."

*“Return any drinker, s.t. there does not exist any other drinker, s.t. there does not exist any beer liked by that other drinker that is not also liked by the returned drinker and there does not exist any beer liked by the returned drinker that is not also liked by the same other drinker.”*

*Let  $x$  be a drinker, and  $S(x)$  be the set of liked beers by drinker  $x$ .*

*Find any drinker  $x$ , s.t. there does not exist another drinker  $x'$ ,  $x$  for which:*

*$S(x') \subseteq S(x)$  and  $S(x') \supseteq S(x)$*

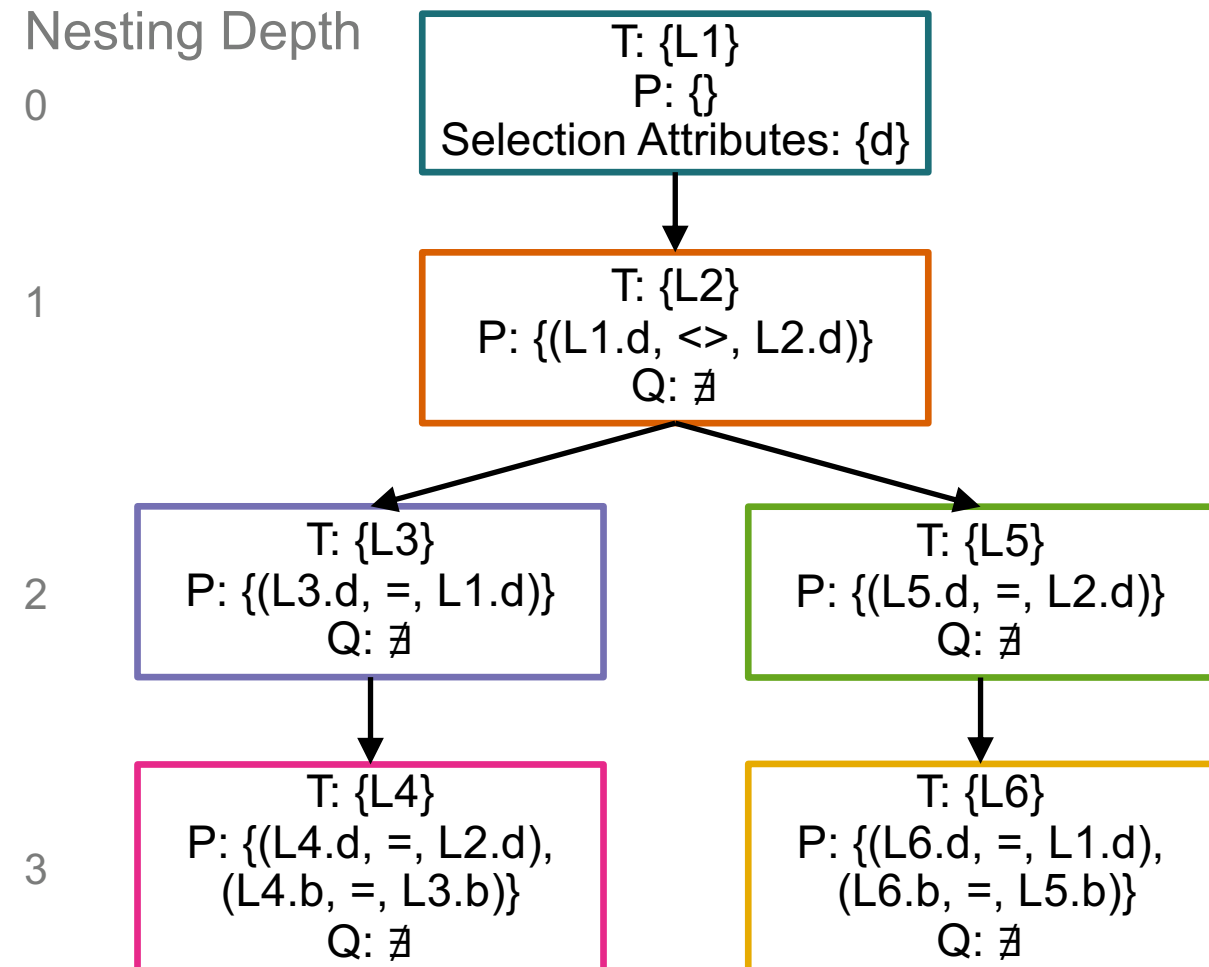
Unique set query: "Find drinkers that like a unique set of beers."

Likes	Likes
drinker	d
beer	b

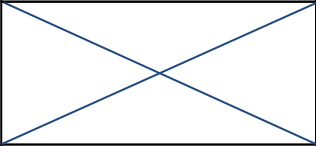


$$\{ L1.d \mid \exists L1 \in \text{Likes} \wedge \\ \nexists L2 \in \text{Likes} [L2.d \neq L1.d \wedge \\ \nexists L3 \in \text{Likes} [L3.d = L1.d \wedge \\ \nexists L4 \in \text{Likes} [L4.d = L2.d \wedge L4.b = L3.b]] \wedge \\ \nexists L5 \in \text{Likes} [L5.d = L2.d \wedge \\ \nexists L6 \in \text{Likes} [L6.d = L1.d \wedge L6.b = L5.b]]]] \}$$

Notice how the logic tree portrays the nesting hierarchy shown in the FOL (TRC) representation of the SQL query.

Each node in the LT represents the root of a scope in the FOL representation. The predicates in each node are the predicates in the root of the scope of a given node (thus the predicates which do not use any additionally quantified variables).



# Atomic predicate classification

		type	
		selection p.	join p.
scope	local (all C are local)	C O V	C O C
	connecting (one C is local, another one is foreign)		C O C
	foreign (all C are foreign)		

Our simple rule: **every predicate needs to have at least one local table identifier.**

Allowed:

- local op value (local selection pred.)
- local op local (local join pred.)
- local op ancestor (connecting join pred.)

Not allowed:

- ancestor op value (foreign selection pred.)
- ancestor op ancestor (foreign join pred.)

# Focus: one single nesting level

- We first restrict ourselves to
  - equi-joins (no inequalities like  $T.A < T.B$ )
  - paths (no siblings = every node can have only one nested child)
  - one single nesting level
  - Boolean queries
  - no foreign predicates
  - only binary relations (thus can be represented as graphs)
  - only one single relation R
  - (and as before only conjunctions)
- Given two such queries, what is a generalization of the homomorphism procedure that works for that fragment?

# Simplifying notation

Schema: R(A,B)

What will become handy, is a short convenient notation for queries

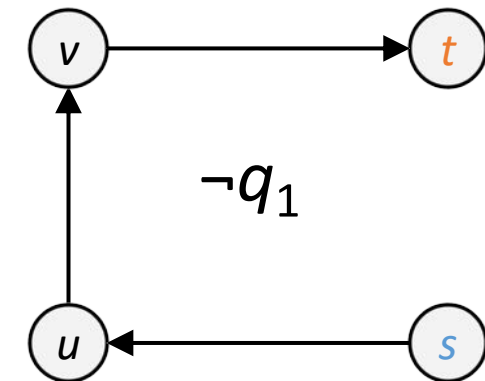
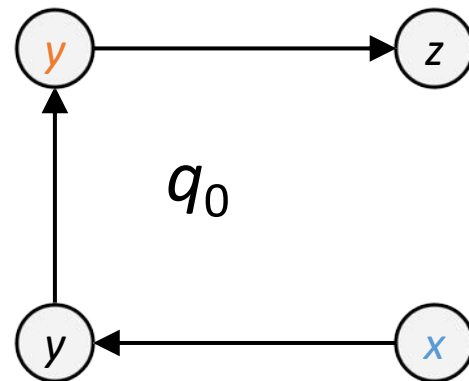
```
SELECT TRUE
FROM R R1, R R2, R R3
WHERE R1.B = R2.A
AND R2.B = R3.A
NOT EXISTS
  (SELECT *
   FROM R R4, R R5, R R6
   WHERE R4.B = R5.A
   AND R5.B = R6.A
   AND R4.A = R1.A
   AND R6.A = R2.B)
```

$q_0 :- R(x,y), R(y,z), R(z,w)$

$q_1(s,t) :- R(s,u), R(u,v), R(v,t), s=x, t=y$

$q :- R(x,y), R(y,z), R(z,w), \neg q_1(x,z)$

$\exists R1, R2, R3 \in R$   
 $(R1.B=R2.A \wedge R2.B=R3.A \wedge$   
 $\nexists R4, R5, R6 \in R$   
 $(R4.B=R5.A \wedge R5.B=R6.A \wedge$   
 $R4.A=R1.A \wedge R6.A = R2.B)$   
 $)$



$s=x, t=y$

# Simplifying notation

Schema: R(A,B)

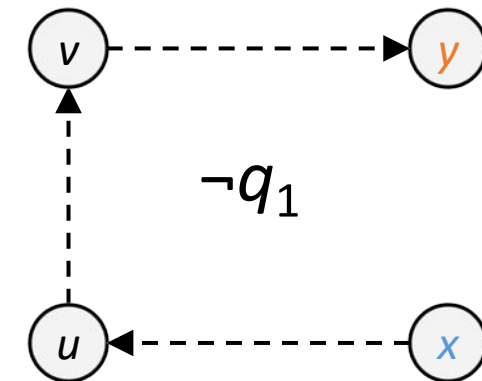
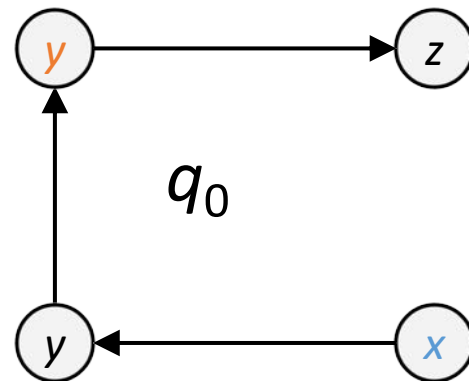
What will become handy, is a short convenient notation for queries

```
SELECT TRUE
FROM R R1, R R2, R R3
WHERE R1.B = R2.A
AND R2.B = R3.A
NOT EXISTS
  (SELECT *
   FROM R R4, R R5, R R6
   WHERE R4.B = R5.A
   AND R5.B = R6.A
   AND R4.A = R1.A
   AND R6.A = R2.B)
```

$q_0 := R(x,y), R(y,z), R(z,w)$

$\neg q_1 := R(x,u), R(u,v), R(v,y)$

$\exists R1, R2, R3 \in R$   
 $(R1.B=R2.A \wedge R2.B=R3.A \wedge$   
 $\nexists R4, R5, R6 \in R$   
 $(R4.B=R5.A \wedge R5.B=R6.A \wedge$   
 $R4.A=R1.A \wedge R6.A = R2.B)$   
 $)$





# Simplifying notation

Schema: R(A,B)

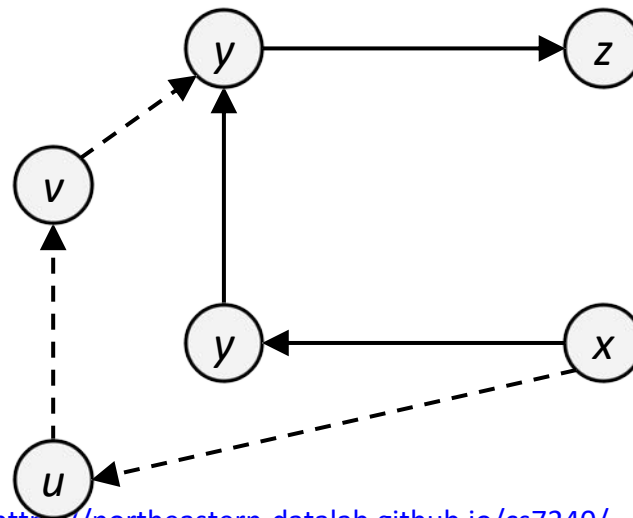
What will become handy, is a short convenient notation for queries

```
SELECT TRUE
FROM R R1, R R2, R R3
WHERE R1.B = R2.A
AND R2.B = R3.A
NOT EXISTS
  (SELECT *
   FROM R R4, R R5, R R6
   WHERE R4.B = R5.A
   AND R5.B = R6.A
   AND R4.A = R1.A
   AND R6.A = R2.B)
```

$q_0 :- R(x,y), R(y,z), R(z,w)$

$\neg q_1 :- R(x,u), R(u,v), R(v,y)$

$\exists R1, R2, R3 \in R$   
 $(R1.B=R2.A \wedge R2.B=R3.A \wedge$   
 $\nexists R4, R5, R6 \in R$   
 $(R4.B=R5.A \wedge R5.B=R6.A \wedge$   
 $R4.A=R1.A \wedge R6.A = R2.B)$   
 $)$



*Cartesian product:  $R'(x,y,z,w) = R(x,y), R(y,z), R(z,w)$   
can be expressed in guarded  
fragment of FOL (with negation)?  
But single join already not guarded*

*See Barany, Cate, Segoufin,  
"Guarded negatation", JACM 2015*

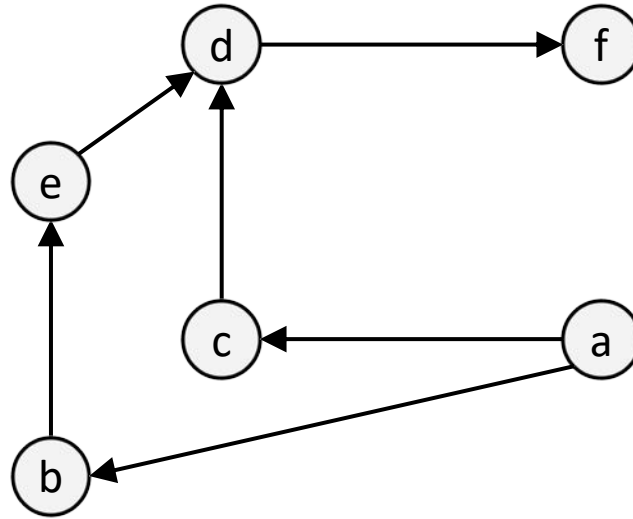
*guardedness*

# Exercise

Schema:  $R(A,B)$

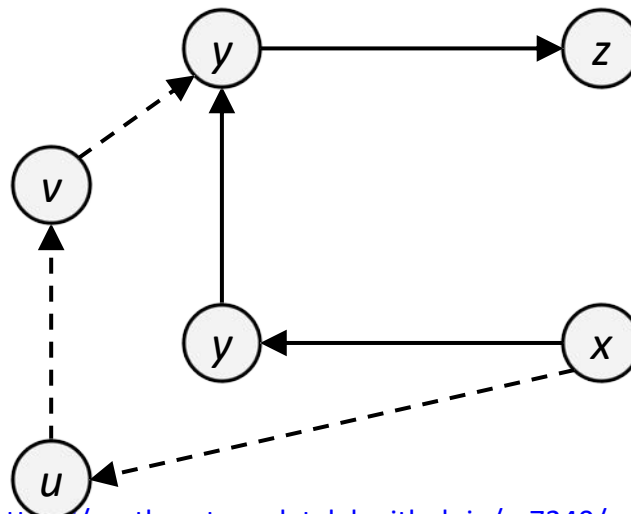


Database  $D$



Does the query below evaluate to true on above database?

Query  $q$

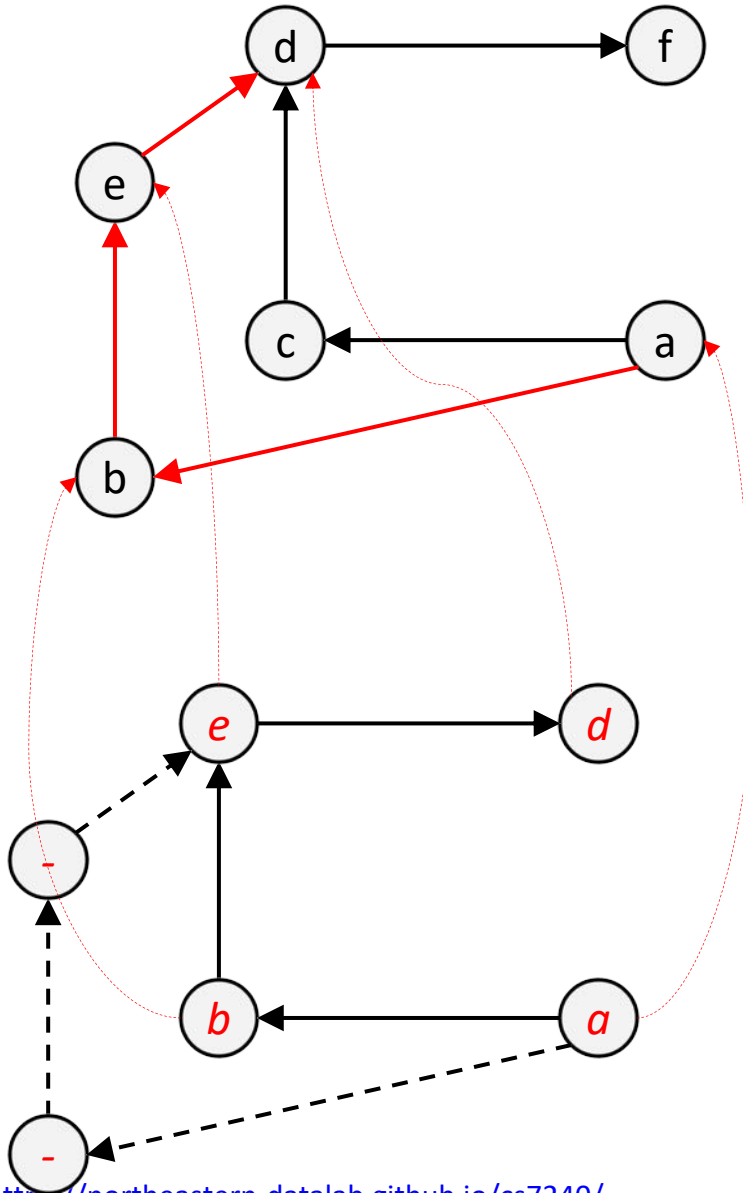


# Exercise

Schema:  $R(A,B)$



Database  $D$



Query  $q$

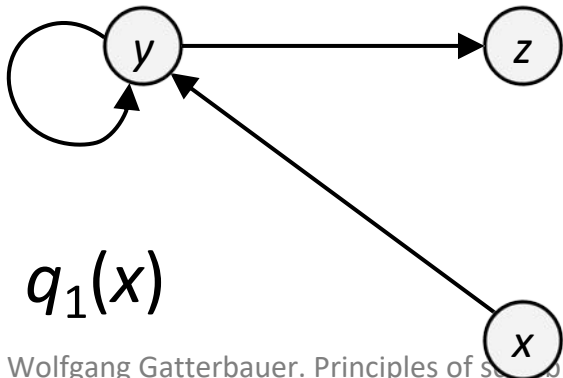
# Question

- Find two such nested queries (somehow leveraging the example below) that are equivalent (based on some simple reasoning)
- What is then the \*structured\* procedure to prove equivalence?

## Example

$q_1(x) :- R(x,y), R(y,y), R(y,z)$

$q_2(s) :- R(s,u), R(u,w), R(s,v), R(u,w), R(u,v), R(v,v)$

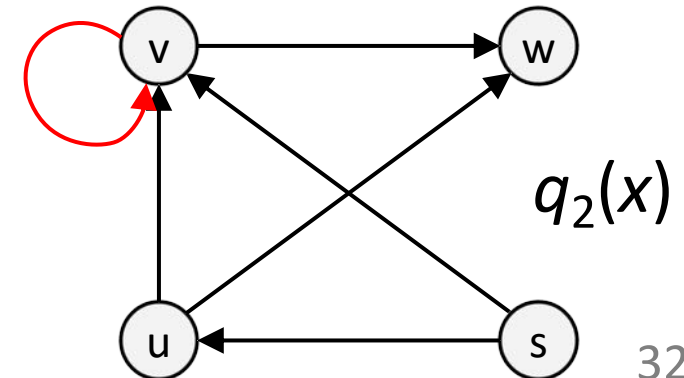


~~$h_{1 \rightarrow 2}: \{(x,s), (y,v), (z,w)\}$~~

$h_{2 \rightarrow 1}: \{(s,x), (u,y), (v,y), (w,z)\}$

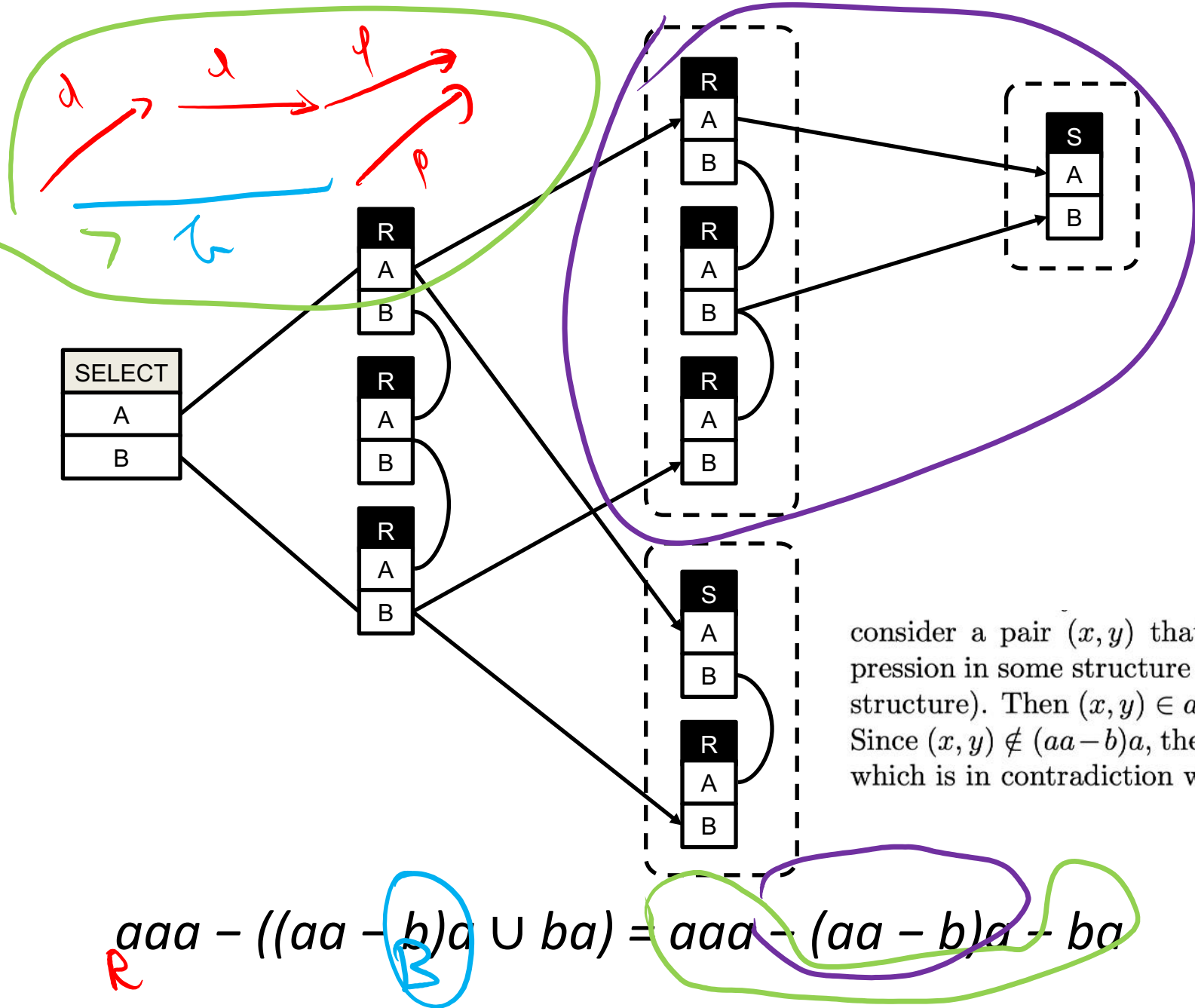
$q_1 \not\subseteq q_2$

$q_1 \subseteq q_2$



# Undecidability ☹️

- Unfortunately, the following problem is already undecidable
  - Consider the class of nested queries with maximal nesting level 2, no disjunctions, our safety restrictions from earlier, set semantics, arbitrary number of siblings
  - Deciding whether any given query is finitely satisfiable is undecidable.
- This follows non-trivially from the following Arxiv paper:
  - **“Undecidability of satisfiability in the algebra of finite binary relations with union, composition, and difference”** by Tony Tan, Jan Van den Bussche, Xiaowang Zhang, Corr 1406.0349.  
<https://arxiv.org/abs/1406.0349>



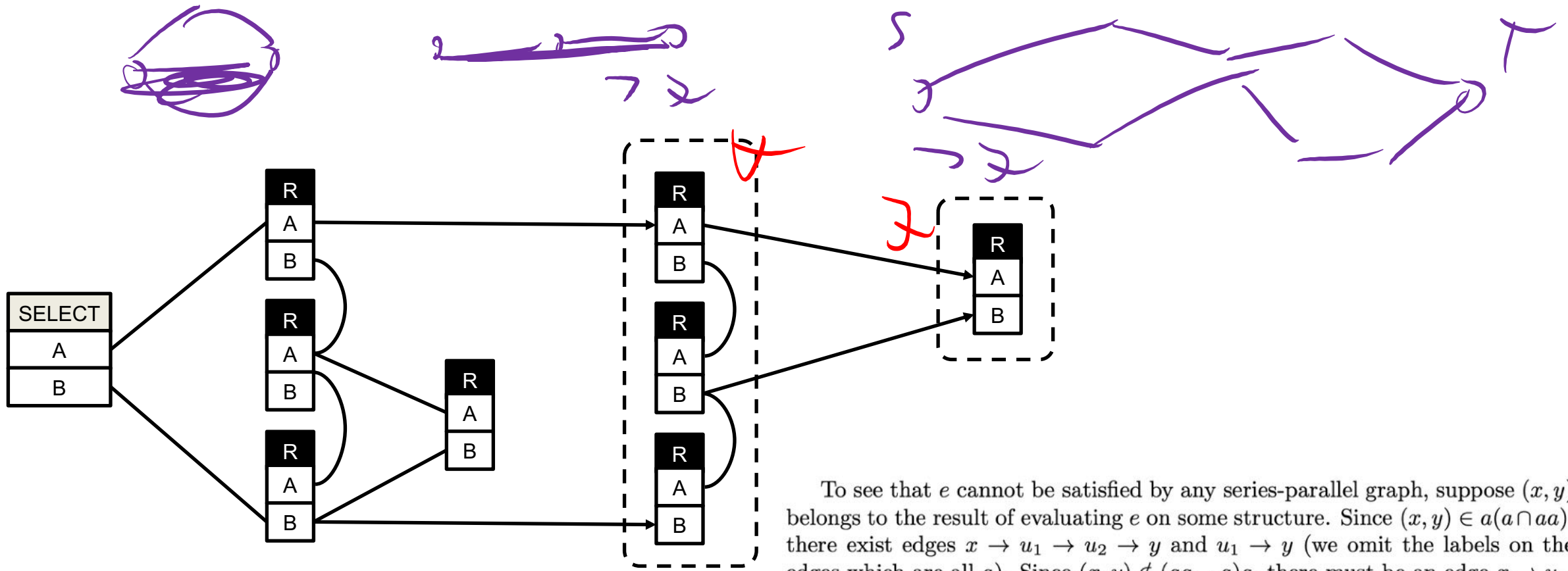
$$\begin{aligned}
 a &\rightarrow R(A, B) \\
 b &\rightarrow S(A, B)
 \end{aligned}$$

$$\begin{aligned}
 &= aaa - (aa - b)a - ba \\
 &= aef - (ae - b)f - bf \\
 &= aef - aef \cup bf - bf
 \end{aligned}$$

consider a pair  $(x, y)$  that would belong to the result of evaluating this expression in some structure (for brevity we are omitting explicit reference to this structure). Then  $(x, y) \in aaa$  so there exist  $a$ -edges  $(x, x_1)$ ,  $(x_1, x_2)$ , and  $(x_2, y)$ . Since  $(x, y) \notin (aa - b)a$ , the  $b$ -edge  $(x, x_2)$  must be present. But then  $(x, y) \in ba$ , which is in contradiction with the last part of the expression.  $\square$

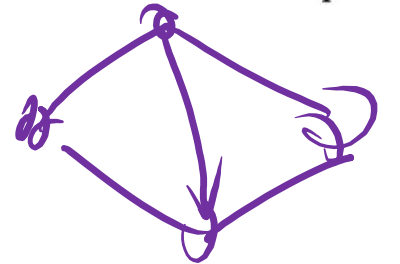
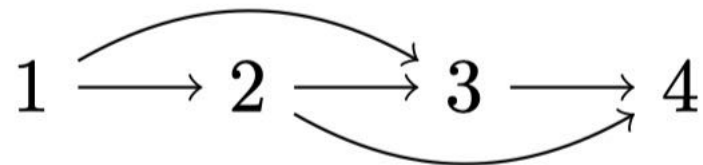
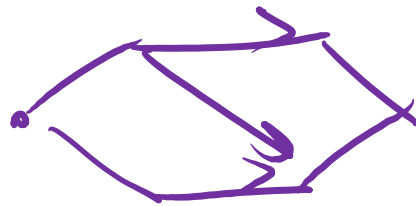
$$\overset{R}{aaa} - ((aa - \overset{B}{b})a \cup ba) = aaa - (aa - b)a - ba$$

$$X - (Y \cup Z) = X - Y - Z$$



To see that  $e$  cannot be satisfied by any series-parallel graph, suppose  $(x, y)$  belongs to the result of evaluating  $e$  on some structure. Since  $(x, y) \in a(a \cap aa)$ , there exist edges  $x \rightarrow u_1 \rightarrow u_2 \rightarrow y$  and  $u_1 \rightarrow y$  (we omit the labels on the edges which are all  $a$ ). Since  $(x, y) \notin (aa - a)a$ , there must be an edge  $x \rightarrow u_2$ . If at least two of the four elements  $x, u_1, u_2$  and  $y$  are identical, the graph contains a cycle and is not series-parallel. If all four elements are distinct, we have a subgraph isomorphic to  $W$  above, so the structure is not series-parallel

$$a(aa \cap a) - (aa - a)a$$



# Open question



(SIBLINGS) ~ OUTDOOR etc

QUESTIONS

1                      2                      3+

0	CO	—	—
1			?
2			↙
3+	<del>✓</del> ✓	✓	✓