Updated 2020/2/9

T2: Complexity of Query Evaluation
L8: Query containment & Homomorphisms

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp20)

https://northeastern-datalab.github.io/cs7240/sp20/

Date: 2020/1/31

Three Fundamental Algorithmic Problems about Queries

Let L be a database query language.

- The Query Evaluation Problem:
  - Given a query q in L and a database instance D, evaluate q(D)
  - That's the main problem in query processing.

- The Query Equivalence Problem:
  - Given two queries q and q' in L, is it the case that q ≡ q' ?
    - i.e., is it the case that, <u>for every database</u> instance D, we have that q(D) = q'(D)?
  - This problem underlies query processing and optimization, as we often need to transform a given query to an equivalent one.

- The Query Containment Problem:
  - Given two queries q and q' in L, is it the case that q ⊆ q' ?

Source: Phokion Kolaitis

Outline: Complexity of Query Equivalence

- Query equivalence and query containment
  - Graph homomorphisms
  - Homomorphism beyond graphs
  - CQ containment
  - Beyond CQs
  - CQ equivalence under bag semantics
  - CQ minimization
  - Nested queries
  - Tree pattern queries

Why bother about Query Containment

- The Query Containment Problem and Query Equivalence Problem are closely related to each other:
  - $q \equiv q'$ if and only if
    - $q \subseteq q'$ and $q' \subseteq q$
  - $q \subseteq q'$ if and only if
    - $q \equiv (q \cap q')$

Complexity of Equivalence and Containment

- Theorem: The Query Equivalence Problem for relational calculus queries is...

  ... undecidable ☹

- Proof: Use Trakhtenbrot's Theorem (1949):
  - The Finite Validity Problem (problem of validity in FOL on the class of all finite models) is undecidable.
  - Finite Validity Problem $\preccurlyeq$ Query Equivalence Problem
    - If ψ^* is a fixed finitely valid relational calculus sentence, then for every relational calculus sentence φ, we have that: φ is finitely valid $\Leftrightarrow \varphi \equiv \psi^*$.
- Corollary: The Query Containment Problem for relational calculus queries in undecidable.
  - Proof: Query Equivalence $\preccurlyeq$ Query Containment, since

    $q \equiv q' \Leftrightarrow q \subseteq q'$ and $q' \subseteq q$.

Source: Phokion Kolaitis

Complexity of the Query Evaluation Problem

- The Query Evaluation Problem for Relational Calculus:
  - Given a RC formula φ and a database instance D, find $\varphi^{adom}(D)$.

- Theorem: The Query Evaluation Problem for Relational Calculus is ...

  ... PSPACE-complete.
  - PSPACE: decision problems, can be solved using an amount of memory that is polynomial in the input length (~ in polynomial amount of space).
  - PSPACE-complete: PSPACE + every other PSPACE problem can be transformed to it in polynomial time (PSPACE-hard)
- Proof: We need to show both
  - This problem is in PSPACE.
  - This problem is PSPACE-hard. (We only focus on this task)

Source: Phokion Kolaitis

Complexity of the Query Evaluation Problem

- Theorem: The Query Evaluation Problem for Relational Calculus is PSPACE-hard.
- Reduction uses QBF – Quantified Boolean Formulas
  - Given QBF $\forall x_1 \exists x_2 \ldots \forall x_k \psi$,
  - is it true or false (notice every variable is quantified = bound at beginning of sentence, there are no free variables)
- Proof
  - Show that QBF $\preccurlyeq p$ Query Evaluation for Relational Calculus

Source: Phokion Kolaitis

Complexity of the Query Evaluation Problem

Proof: Show that QBF $\preccurlyeq p$ Query Evaluation for Relational Calculus

- Given QBF $\forall x_1 \exists x_2 \ldots \forall x_k \psi$,
- Let V and P be two unary relation symbols
- Obtain ψ^* from ψ by replacing x_i by $P(x_i)$, and $\neg x_i$ by $\neg P(x_i)$
- Let D be the database instance with V = {0,1}, P={1}.
- Then the following statements are equivalent:
  - $\forall x_1 \exists x_2 \ldots \forall x_k \psi$ is true
  - $\forall x_1 (V(x_1) \rightarrow \exists x_2 (V(x_2) \wedge (\ldots \forall x_k(V(x_k) \rightarrow \psi^*))\ldots)$ is true on D.

Source: Phokion Kolaitis

Sublanguages of Relational Calculus

- Question: Are there interesting sublanguages of relational calculus for which the Query Containment Problem and the Query Evaluation Problem are “easier” than the full relational calculus?

- Answer:
  - Yes, the language of Conjunctive Queries (CQs) is such a sublanguage.
  - Moreover, conjunctive queries are the most frequently asked queries against relational databases.

Source: Phokion Kolaitis

Conjunctive Queries (CQs)

- Definition:
  - A CQ is a query expressible by a RC formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\wedge$ and $\exists$ only.

    $\{ (x_1,\ldots,x_k): \ \exists z_1 \ldots \exists z_m \ \phi(x_1, \ldots,x_k, z_1,\ldots,z_k) \}$,
  - where $\phi(x_1, \ldots,x_k, z_1,\ldots,z_k)$ is a conjunction of atomic formulas of the form $R(y_1,\ldots,y_m)$.
  - Prenex formula: prefix (quantifiers & bound variables), then quantifier-free part
- Equivalently, a CQ is a query expressible by a RA expression of the form
  - $\pi_X(\sigma_\Theta(R_1 \times \ldots \times R_n))$, where
  - Θ is a conjunction of equality atomic formulas (equijoin).
- Equivalently, a CQ is a query expressible by an SQL expression of the form
  - SELECT <list of attributes>
    FROM <list of relation names>
    WHERE <conjunction of equalities>

Source: Phokion Kolaitis

Conjunctive Queries (CQs)

- Definition:
  - A CQ is a query expressible by a RC formula in prenex normal form built from atomic formulas $R(y_1,\ldots,y_n)$, and $\wedge$ and $\exists$ only.

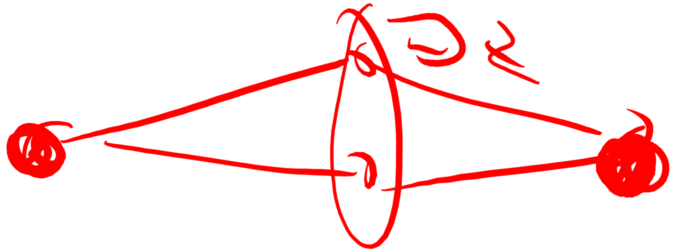
    $\{ (x_1,\ldots,x_k): \exists z_1 \ldots \exists z_m \ \phi(x_1, \ldots,x_k, z_1,\ldots,z_k) \}$,
  - where $\phi(x_1, \ldots,x_k, z_1,\ldots,z_k)$ is a conjunction of atomic formulas of the form $R(y_1,\ldots,y_m)$.

- Equivalently, a CQ can be written as a logic-programming rule:

  $Q(x_1,\ldots,x_k)$:- $R_1(\mathbf{u}_1), \ldots, R_n(\mathbf{u}_n)$, where
  - Each variable x_i occurs in the right-hand side of the rule.
  - Each $\mathbf{u}_i$ is a tuple of variables (not necessarily distinct)
  - The variables occurring in the right-hand side (the body), but not in the left-hand side (the head) of the rule are existentially quantified (but the quantifiers are not displayed).

Source: Phokion Kolaitis

Examples of Conjunctive Queries



- Path of Length 2: (Binary query)

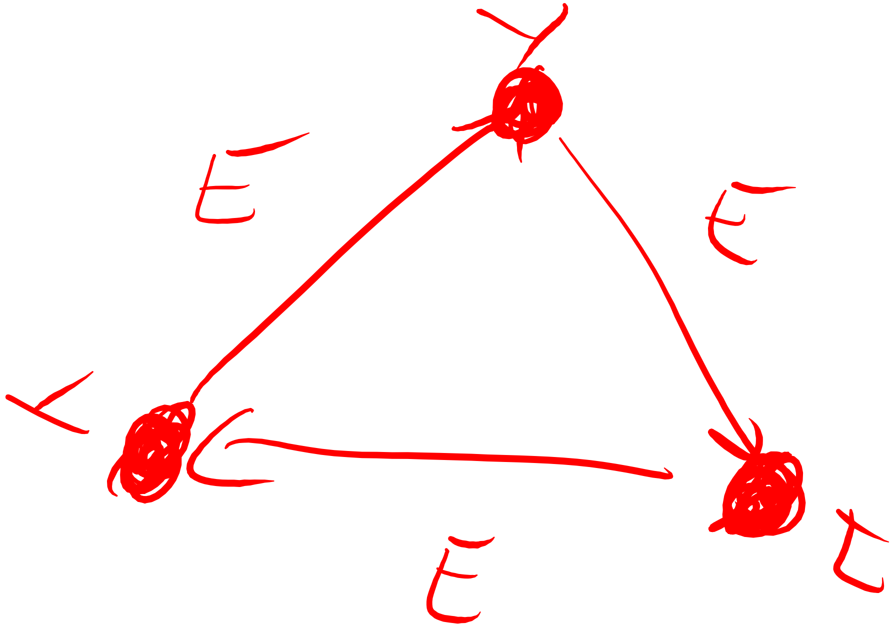
  $\{(x,y): \exists\, z\ (E(x,z) \wedge E(z,y))\}$

  - As a relational algebra expression:

    $\pi_{1,4}(\sigma_{\$2 = \$3}\ (E \times E))$

  - As a Datalogrule:

    q(x,y) :- E(x,z), E(z,y)



- Cycle of Length 3: (Boolean query)

  $\exists x\ \exists y\ \exists z\ (E(x,y) \wedge E(y,z) \wedge E(z,x))$

  - As a rule (the head has no variables)

    Q :- E(x,y), E(y,z), E(z,x)

Conjunctive Queries

- Every natural join is a conjunctive query with ...

  ... no existentially quantified variables
- Example: Given P(A,B,C), R(B,C,D)
  - $P \bowtie R = \{(x,y,z,w)\colon\ P(x,y,z) \wedge R(y,z,w)\}$
  - q(x,y,z,w) :- P(x,y,z), R(y,z,w)

    (no variables are existentially quantified)
  - SELECT P.A, P.B, P.C, R.D
    FROM P, R
    WHERE P.B = R.B AND P.C = R.C
- Conjunctive queries are also known as SPJ-queries (SELECT-PROJECT-JOIN queries)

Conjunctive Query Evaluation and Containment

- Definition: Two fundamental problems about CQs
  - Conjunctive Query Evaluation (CQE):
    - Given a conjunctive query q and an instance D, find q(D).
  - Conjunctive Query Containment (CQC):
    - Given two k-ary conjunctive queries q_1 and q_2, is it true that $q_1 \subseteq q_2$? (i.e., for every instance D, we have that q1(D) ⊆ q2(D))
    - Given two Boolean conjunctive queries q1 and q2, is it true that $q_1 \models q_2$? (that is, for all D, if $D \models q_1$, then $D \models q_2$)?

- Notice that CQC is logical implication.
- Later today: connection to homomorphisms

Vardi's Taxonomy of the Query Evaluation Problem

M.Y Vardi, "The Complexity of Relational Query Languages", 1982

- Definition: Let L be a database query language.
  - The combined complexity of L is the decision problem:
    - given an L-sentence and a database instance D, is φ true on D?
    - In symbols, does $D \vDash \varphi$ (does D satisfy φ)?

  - The data complexity of L is the family of the following decision problems P_{φ}, where φ is an L-sentence:
    - given a database instance D, does $D \vDash \varphi$?

  - The query complexity of L is the family of the following decision problems P_D, where D is a database instance:
    - given an L-sentence φ, does $D \vDash \varphi$?

Vardi's Taxonomy of the Query Evaluation Problem

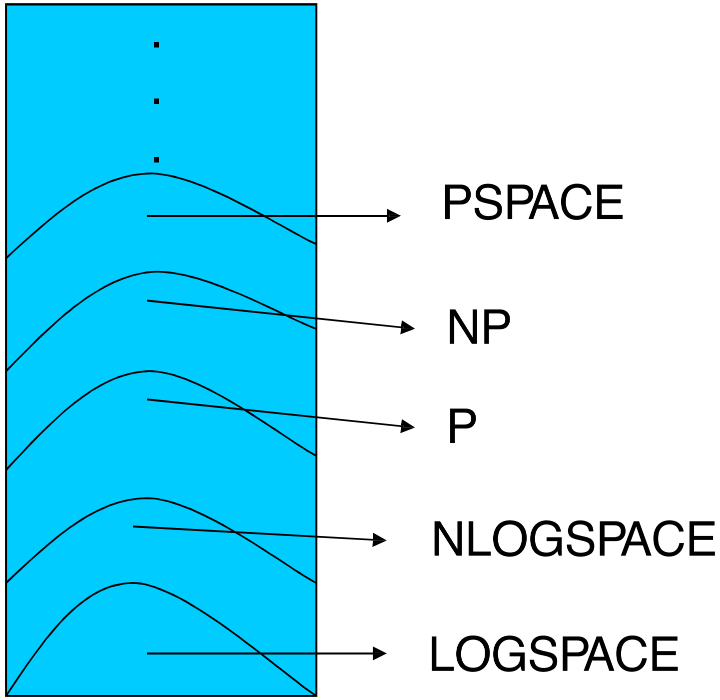
Vardi's "empirical" discovery:

- For most query languages L:
  - The data complexity of L is of lower complexity than both the combined complexity of L and the query complexity of L.
  - The query complexity of L can be as hard as the combined complexity of L.

Source: Phokion Kolaitis

Taxonomy of the Query Evaluation Problem for Relational Calculus

Complexity Classes



The Query Evaluation Problem
for Relational Calculus

Problem	**Complexity**
Combined Complexity	PSPACE-complete
Query Complexity	• in PSPACE • can be PSPACE-complete
Data Complexity	In LOGSPACE

Source: Phokion Kolaitis

Summary

- Relational Algebra and Relational Calculus have "essentially" the same expressive power.

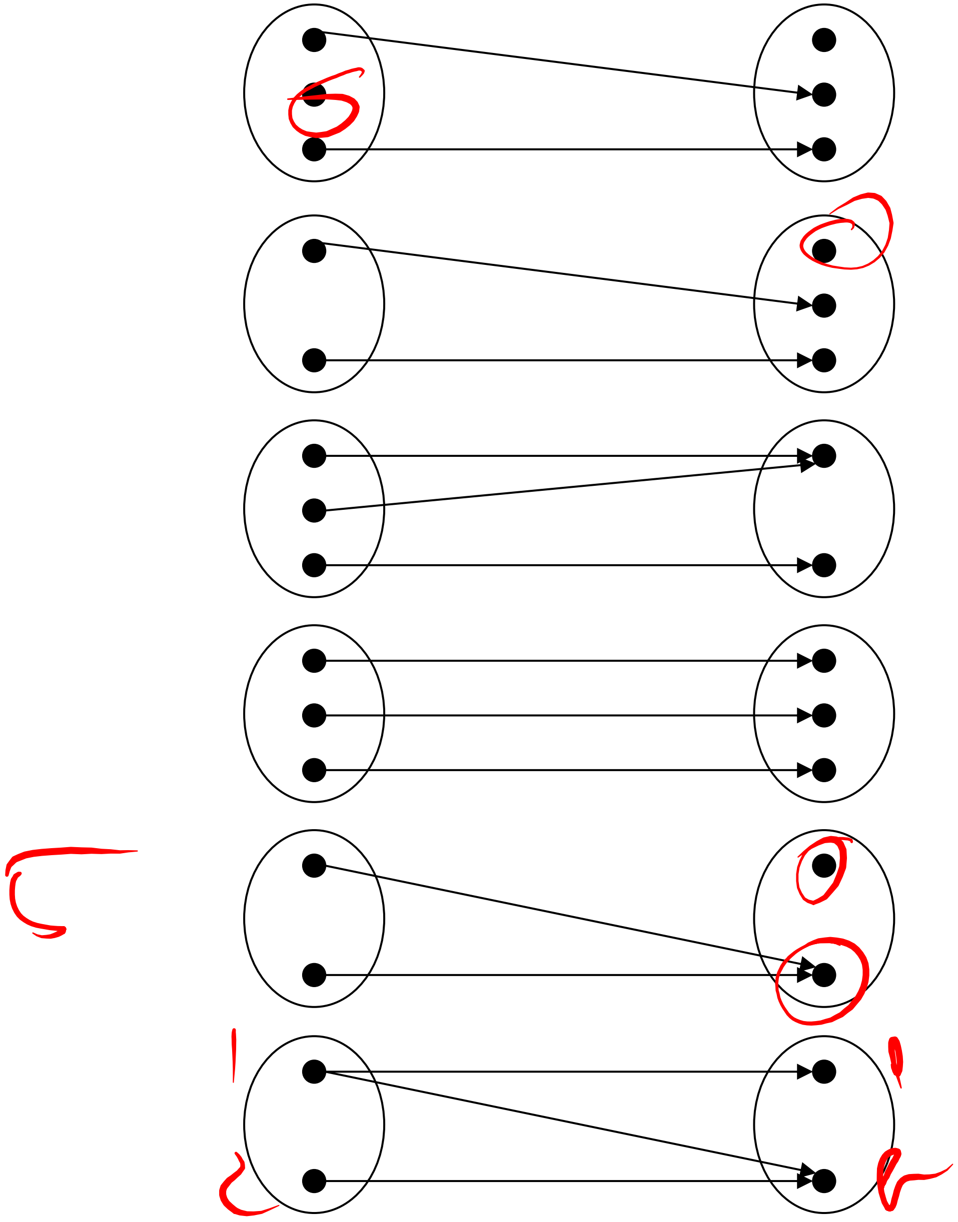
- The Query Equivalence Problem for Relational Calculus is undecidable.
    - Therefore also the Query Containment Problem

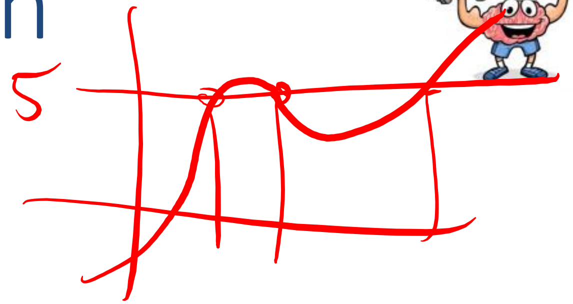
- The Query Evaluation Problem for Relational Calculus:
  - Data Complexity is in LOGSPACE
  - Combined Complexity is PSPACE-complete
  - Query Complexity is PSPACE-complete.

Outline: Complexity of Query Equivalence

- Query equivalence and query containment
  - Graph homomorphisms
  - Homomorphism beyond graphs
  - CQ containment
  - Beyond CQs
  - CQ equivalence under bag semantics
  - CQ minimization
  - Nested queries
  - Tree pattern queries

Mappings: Injection, Surjection, and Bijection





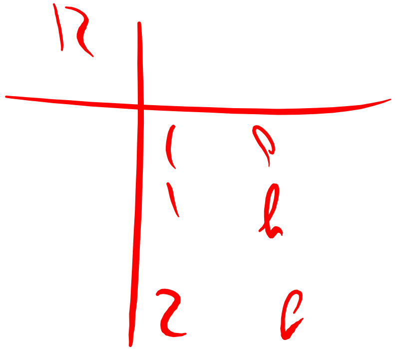
not a mapping (or function)!

injective function (or one-to-one): maps distinct elements of its domain to distinct elements of its codomain

surjective (or onto): every element y in the codomain Y of f has at least one element x in the domain that maps to it

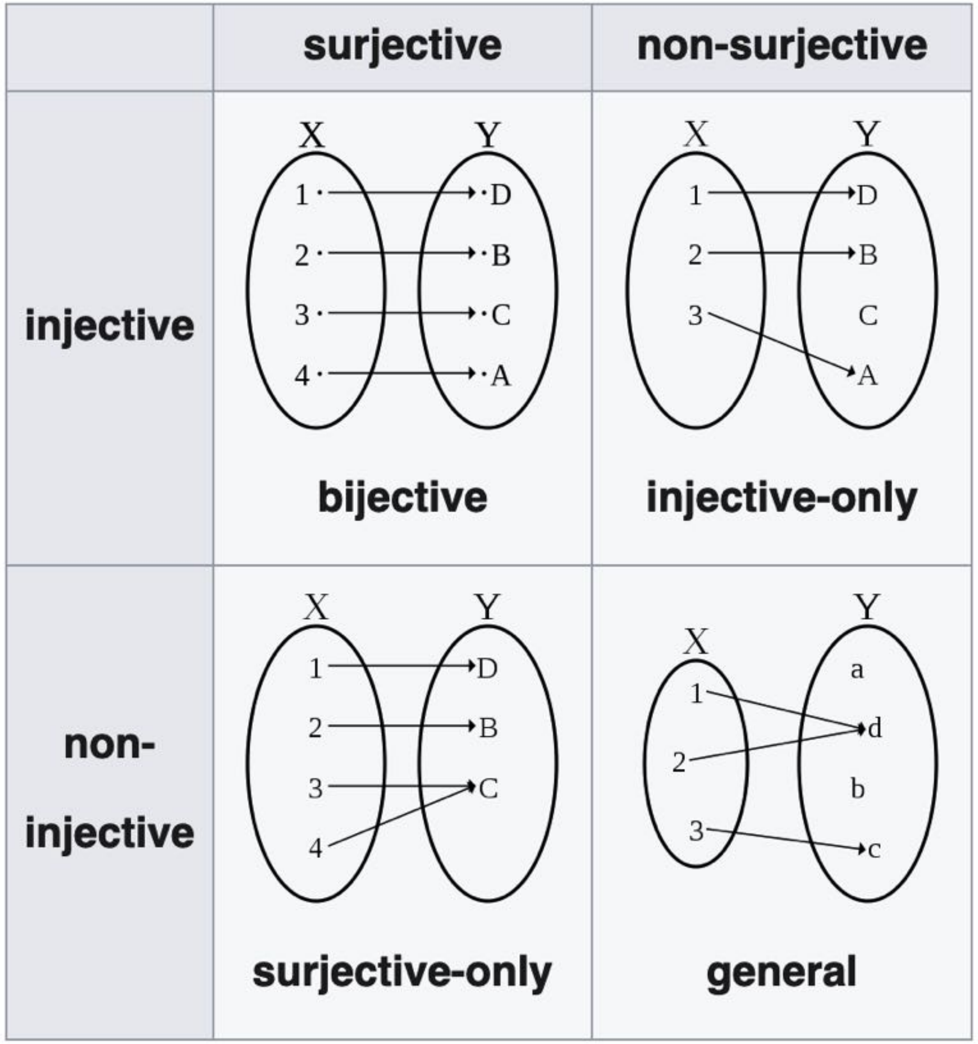
injective & surjective

neighter



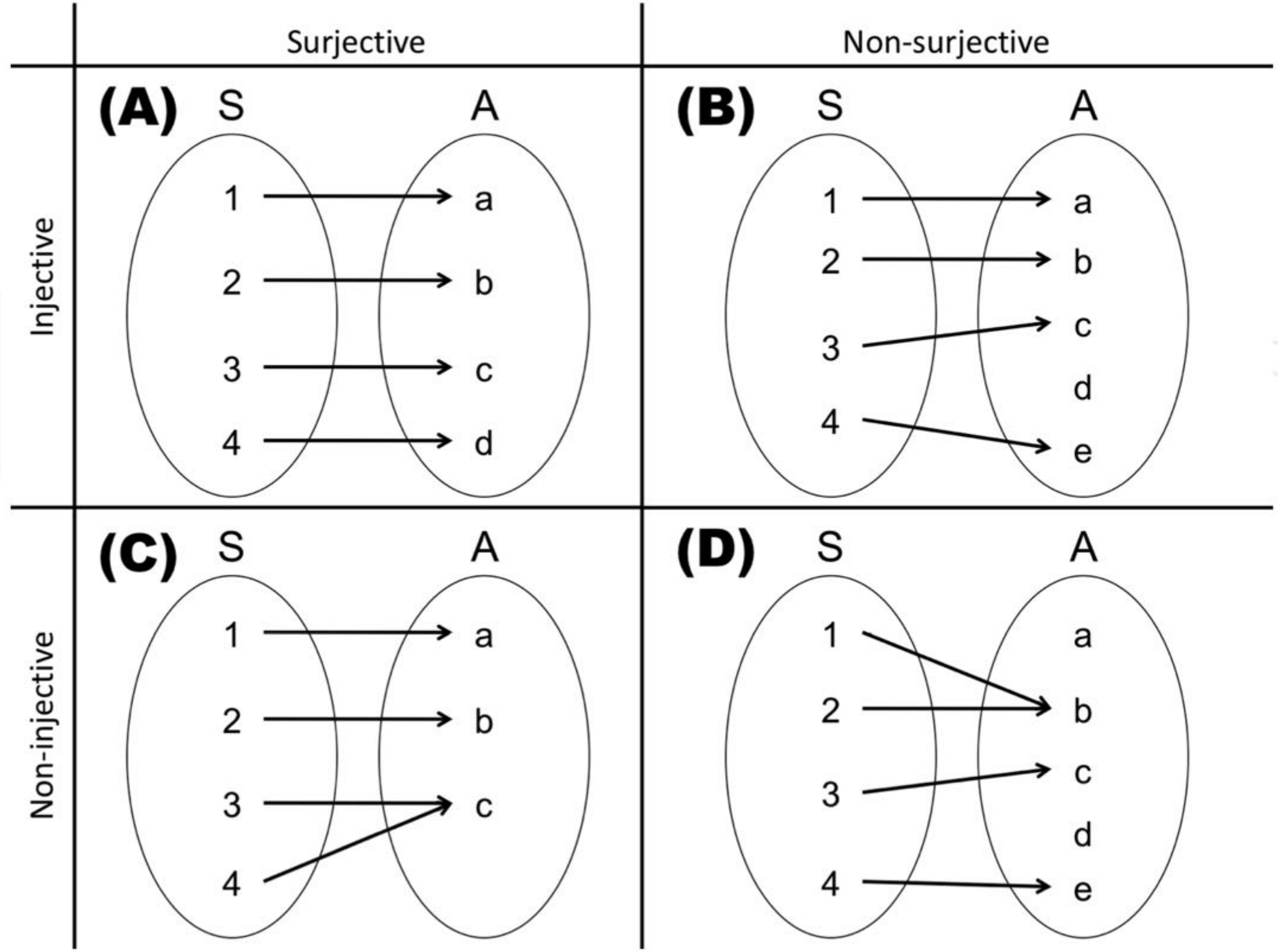
not a mapping

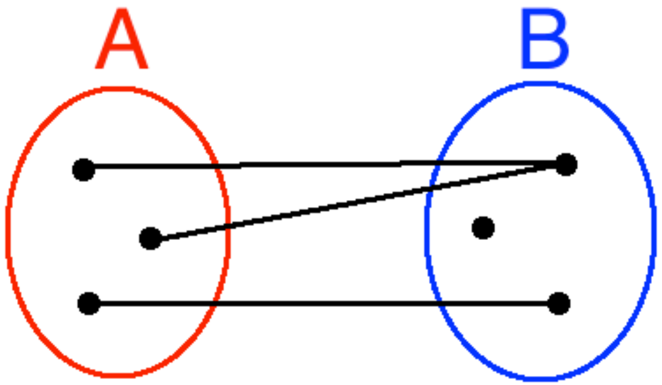
Bijection, Injection, and Surjection



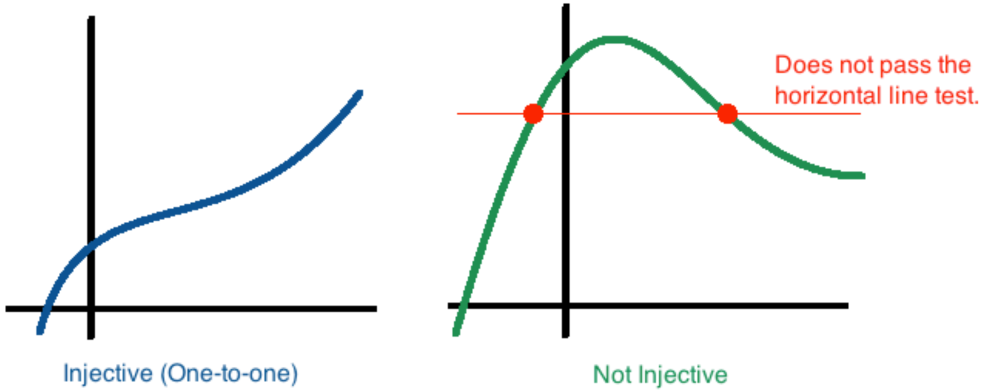
Sources: https://en.wikipedia.org/wiki/Bijection,_injection_and_surjection

Bijection, Injection, and Surjection



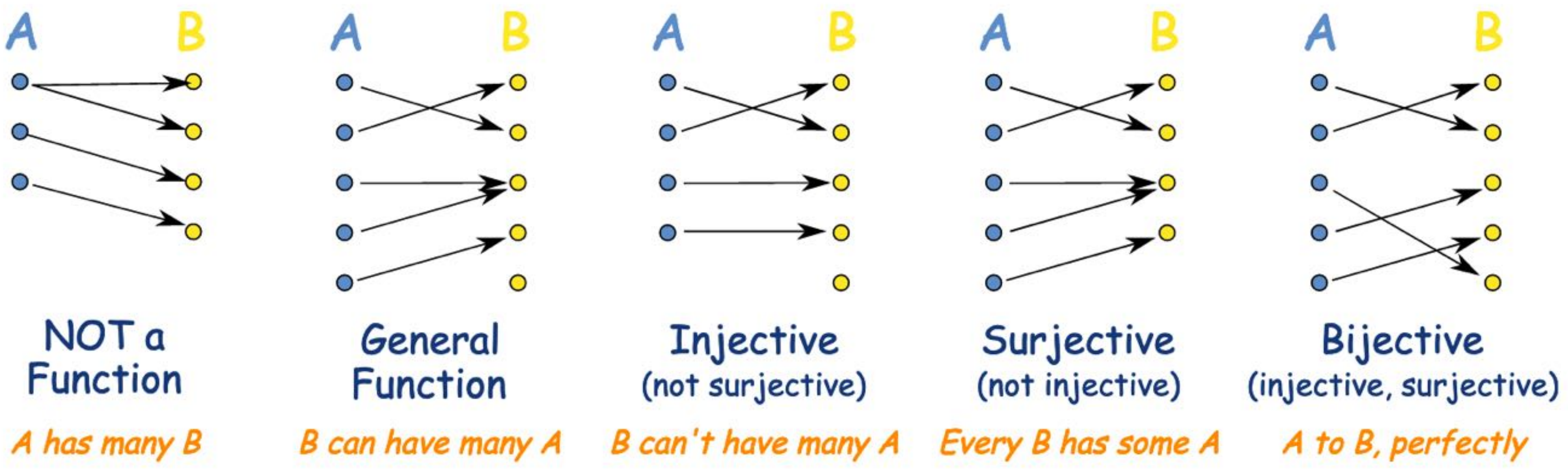


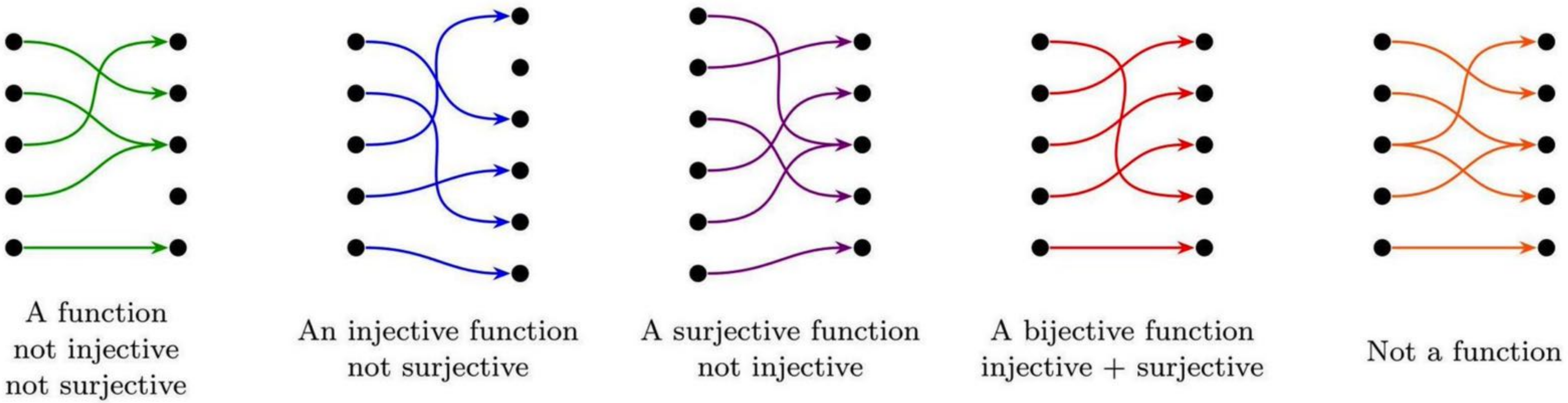
Neither Injective or Surjective
Two elements in set A maps to the same element in set B (not injective), and one element in set B is not in the image or range of the function that maps set A to B (not surjective).



Sources: https://www.intechopen.com/books/protein-interactions/relating-protein-structure-and-function-through-a-bijection-and-its-implications-on-protein-structur, http://mathonline.wikidot.com/injections-surjections-and-bijections

Bijection, Injection, and Surjection





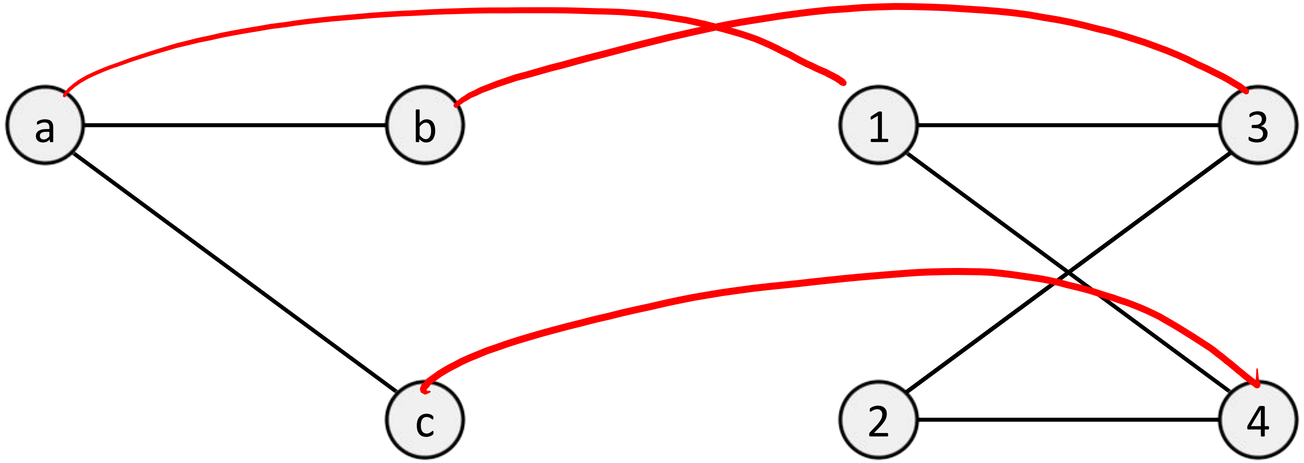
Sources: https://www.mathsisfun.com/sets/injective-surjective-bijective.html, https://twitter.com/jdhamkins/status/841318019397779456,

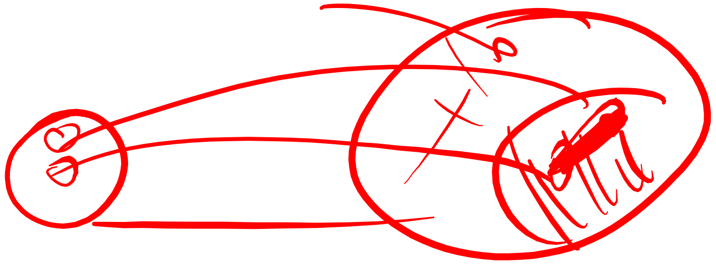
We make a detour to Graph matching

- Finding a correspondence between the nodes and the edges of two graphs that satisfies some (more or less stringent) constraints

Homomorphism

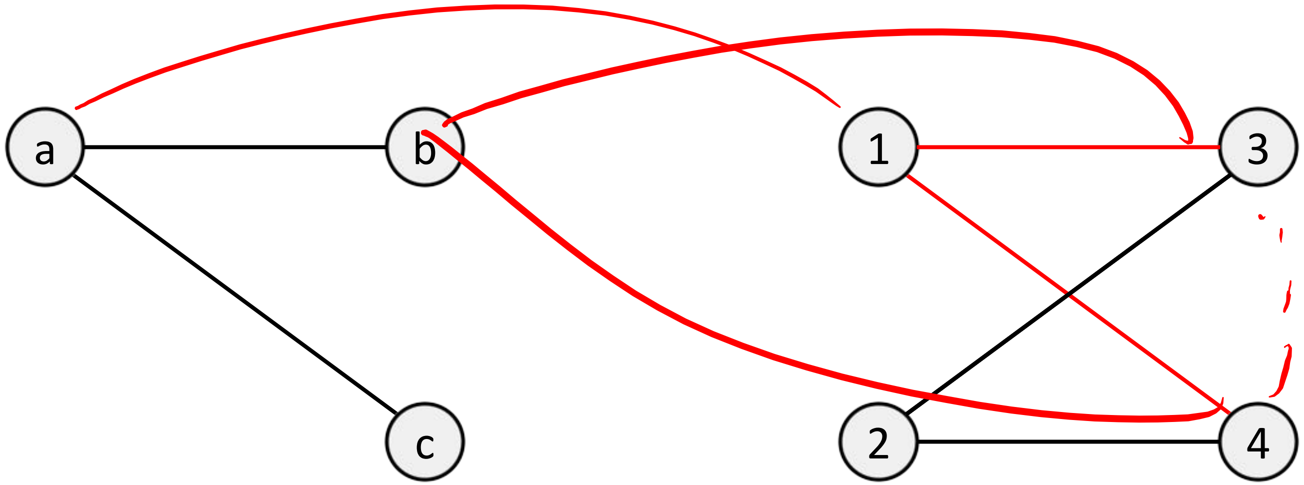
- A graph homomorphism h from graph $G(V_G,E_G)$ to $H(V_H,E_H)$, is a mapping from V_G to V_H such that $\{x,y\} \in E_G$ implies $\{h(x),h(y)\} \in E_H$
  - "edge-preserving": if two nodes in G are linked by an edge, then they are mapped to two nodes in H that are also linked





Homomorphism

- A graph homomorphism *h* from Graph $G(V_G,E_G)$ to $H(V_H,E_H)$, is a mapping from V_G to V_H such that $\{x,y\} \in E_G$ implies $\{h(x),h(y)\} \in E_H$
  - "edge-preserving": if two nodes in *G* are linked by an edge, then they are mapped to two nodes in *H* that are also linked

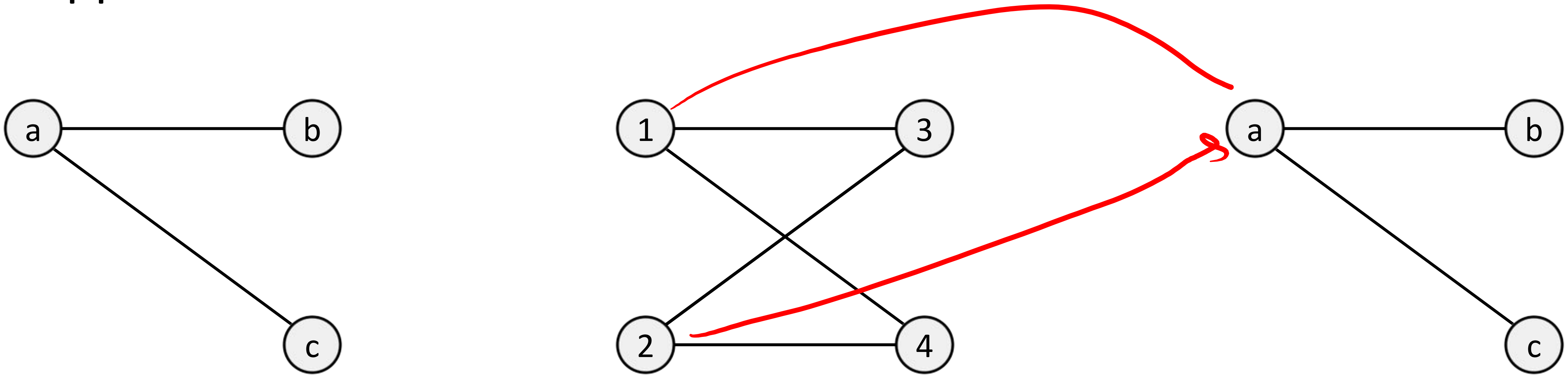


h: {(a,1), (b,3), (c,4)}

does not need to be surjective

Homomorphism

- A graph homomorphism *h* from Graph $G(V_G,E_G)$ to $H(V_H,E_H)$, is a mapping from V_G to V_H such that $\{x,y\} \in E_G$ implies $\{h(x),h(y)\} \in E_H$
  - "edge-preserving": if two nodes in *G* are linked by an edge, then they are mapped to two nodes in *H* that are also linked

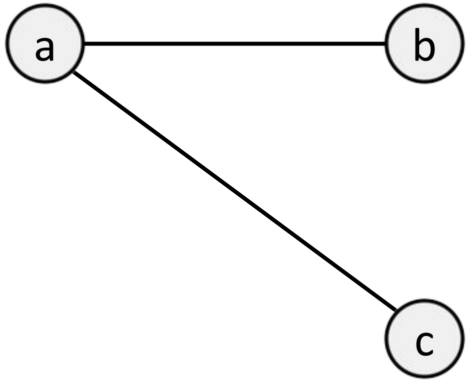


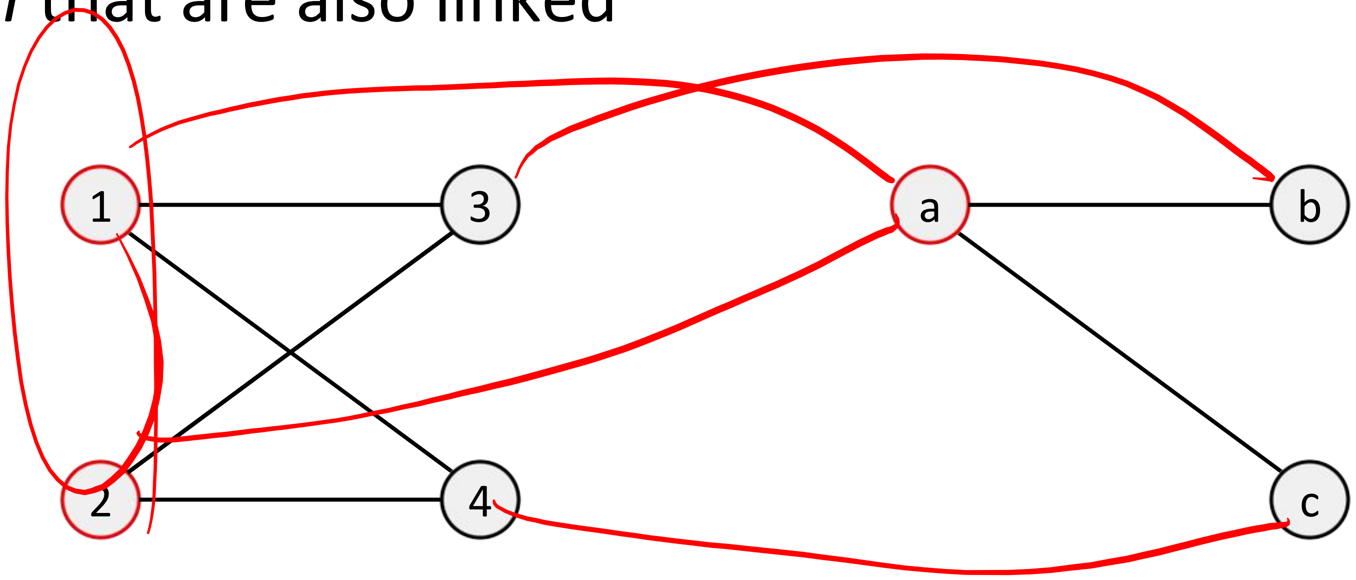
h: {(a,1), (b,3), (c,4)}

does not need to be surjective

Homomorphism

- A graph homomorphism *h* from Graph $G(V_G,E_G)$ to $H(V_H,E_H)$, is a mapping from V_G to V_H such that $\{x,y\} \in E_G$ implies $\{h(x),h(y)\} \in E_H$
  - "edge-preserving": if two nodes in *G* are linked by an edge, then they are mapped to two nodes in *H* that are also linked





h: {(a,1), (b,3), (c,4)}

does not need to be surjective

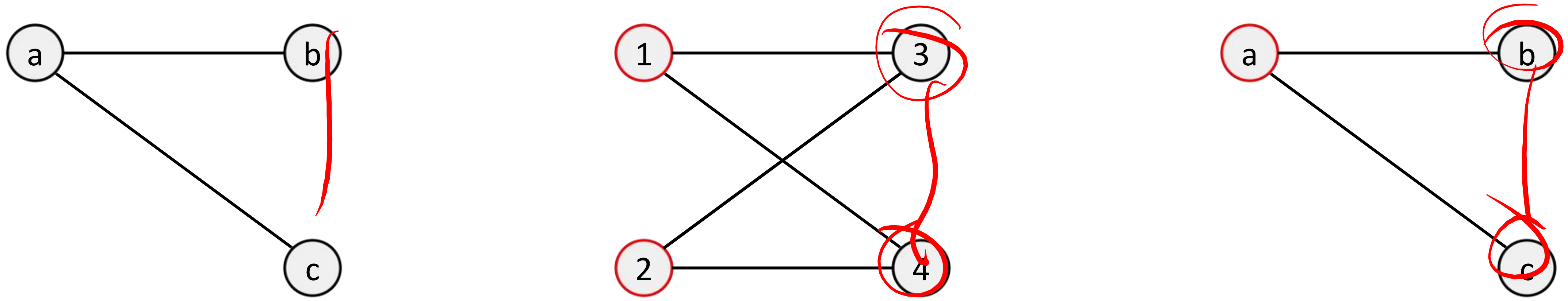
h: {(1,a), (2,a), (3,b), (4,c)}

does not need to be injective

Homomorphism

- A graph homomorphism *h* from Graph $G(V_G,E_G)$ to $H(V_H,E_H)$, is a mapping from V_G to V_H such that $\{x,y\} \in E_G$ implies $\{h(x),h(y)\} \in E_H$
  - "edge-preserving": if two nodes
    mapped to two nodes in *H* that

Correspondence can be many-to-one: nothing prevents that 2 nodes in the first graph are to be mapped to the identical nodes in the second!



h: {(a,1), (b,3), (c,4)}

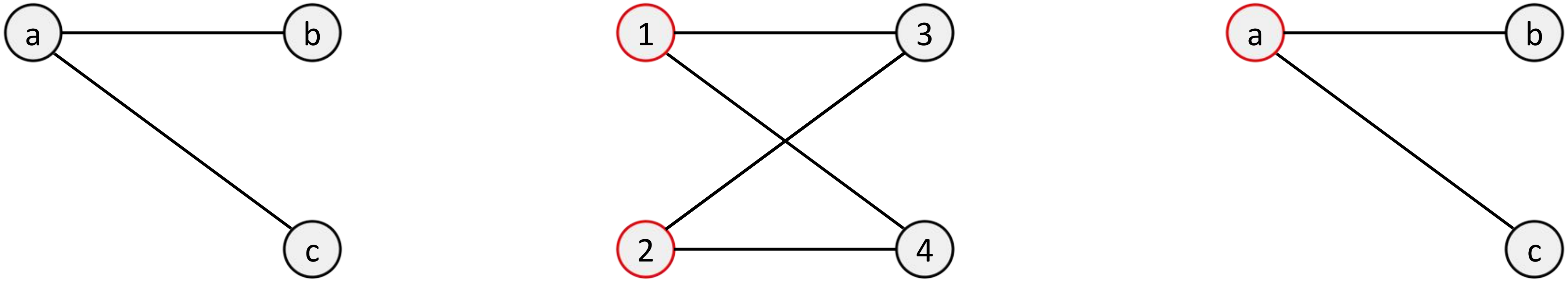
does not need to be surjective

h: {(1,a), (2,a), (3,b), (4,c)}

does not need to be injective

Homomorphism

- A graph homomorphism *h* from Graph $G(V_G,E_G)$ to $H(V_H,E_H)$, is a mapping from V_G to V_H such that $\{x,y\} \in E_G$ implies $\{h(x),h(y)\} \in E_H$
  - "edge-preserving": if two nodes in *G* are linked by an edge, then they are mapped to two nodes in *H* that are also linked



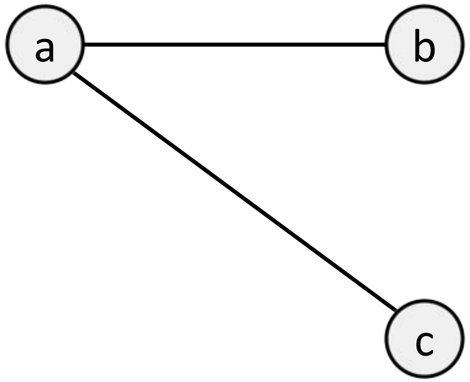
h: {(a,1), (b,3), (c,4)}

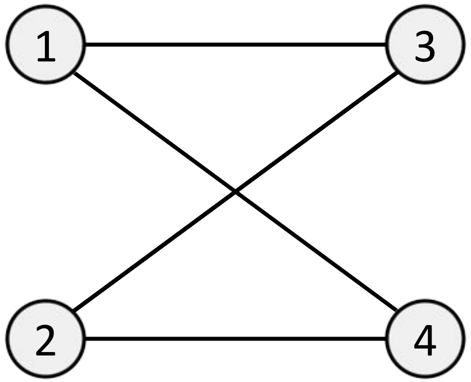
h: {(1,a), (2,a), (3,b), (4,c)}

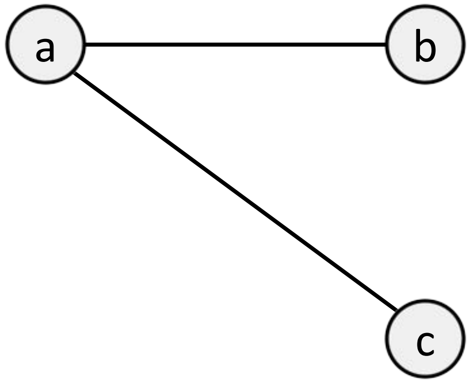
homomorphically equivalent

Graph Isomorphism

- Graphs $G(V_G,E_G)$ and $H(V_H,E_H)$ are isomorphic iff there is an invertible f from V_G to V_H s.t. $\{x,y\} \in E_G$ iff $\{f(u),f(v)\} \in E_H$
  - We need to find a one-to-one correspondence







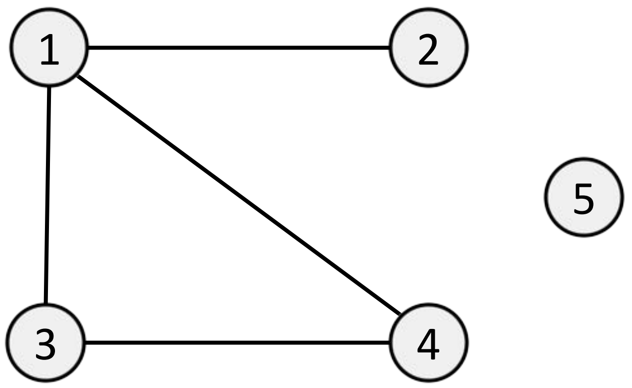
f: {(a,1), (b,3), (c,4)}

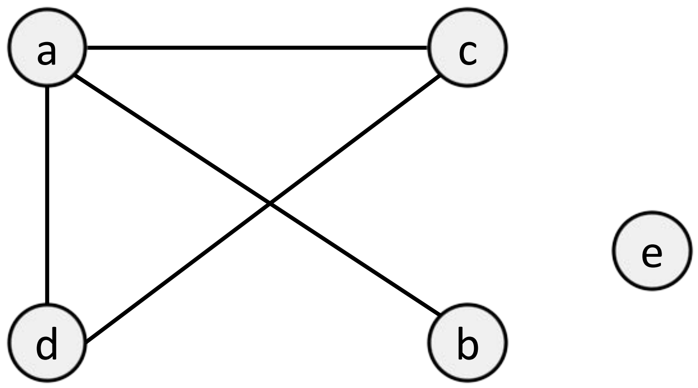
f: {(1,a), (2,a), (3,b), (4,c)}

not possible!

Graph Isomorphism

- Graphs $G(V_G,E_G)$ and $H(V_H,E_H)$ are isomorphic iff there is an <u>invertible</u> f from V_G to V_H s.t. $\{x,y\} \in E_G$ iff $\{f(u),f(v)\} \in E_H$
  - We need to find a one-to-one correspondence





c4 d3

f: {(1,a), (b,2), (c,3), (d,4), (e,5)}

bijection = surjective and injective

Outline: Complexity of Query Equivalence

- Query equivalence and query containment
  - Graph homomorphisms
  - Homomorphism beyond graphs
  - CQ containment
  - Beyond CQs
  - CQ equivalence under bag semantics
  - CQ minimization
  - Nested queries
  - Tree pattern queries

Graph Homomorphism beyond graphs

Definition : *Let G and H be graphs. A homomorphism of G to H is a function f*: $V(G) \to V(H)$ such that

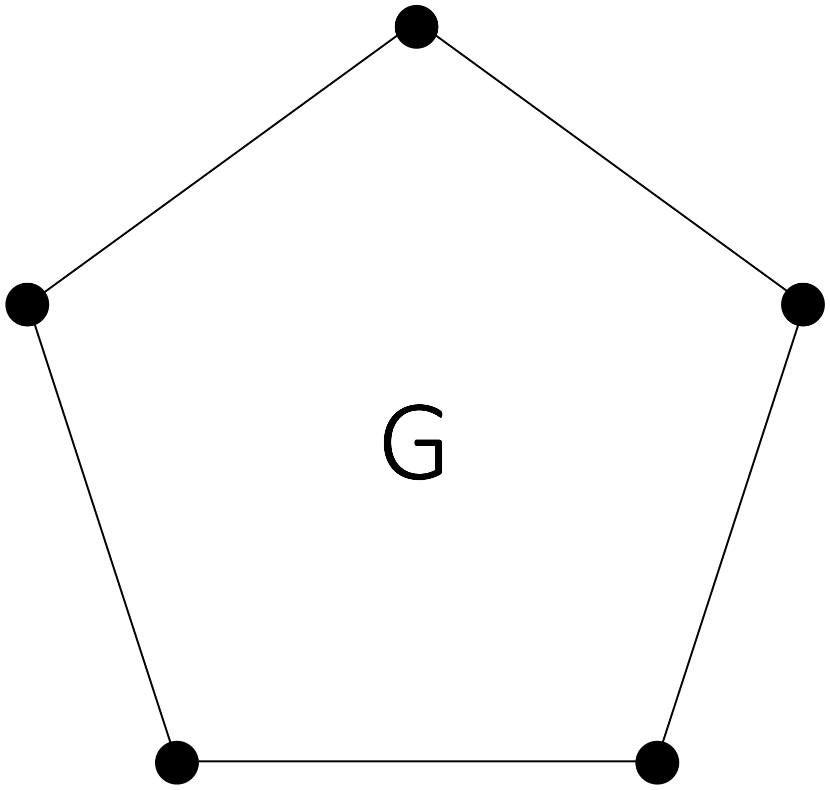
$$(xy) \in E(G) \Rightarrow (f(x), f(y)) \in E(H).$$

We sometimes write $G \to H$ ($G \nrightarrow H$) if there is a homomorphism (no homomorphism) of G to H

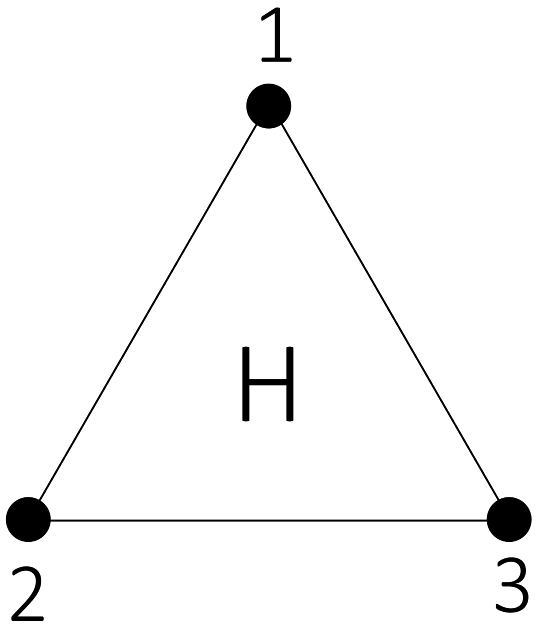
Definition of a homomorphism naturally extends to:

- digraphs
- edge-colored graphs
- relational systems
- constraint satisfaction problems (CSPs)

An example

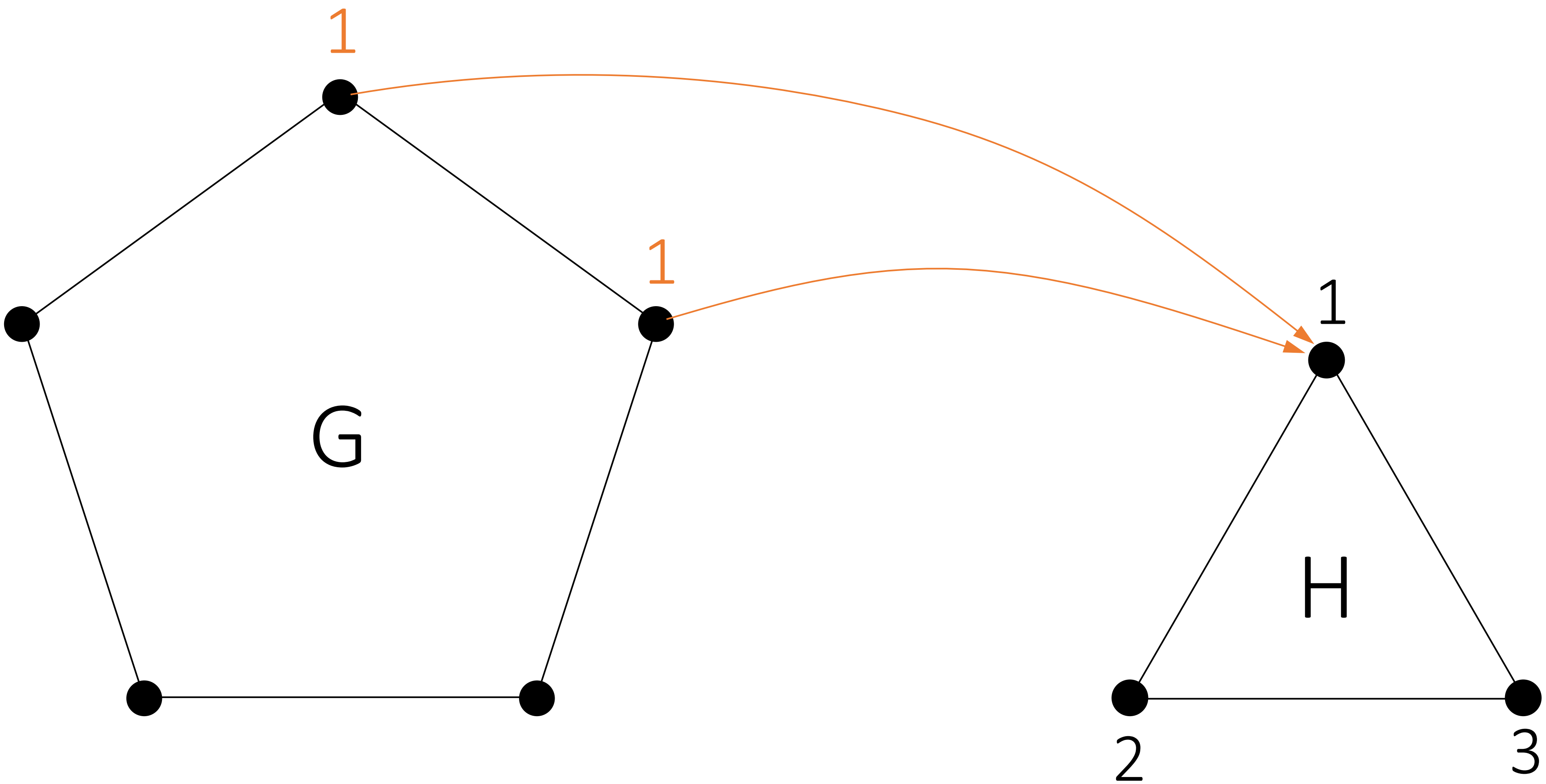


3 "colors" of the vertices



Example by Rick Brewster, Graph homomorphism tutorial, 2006

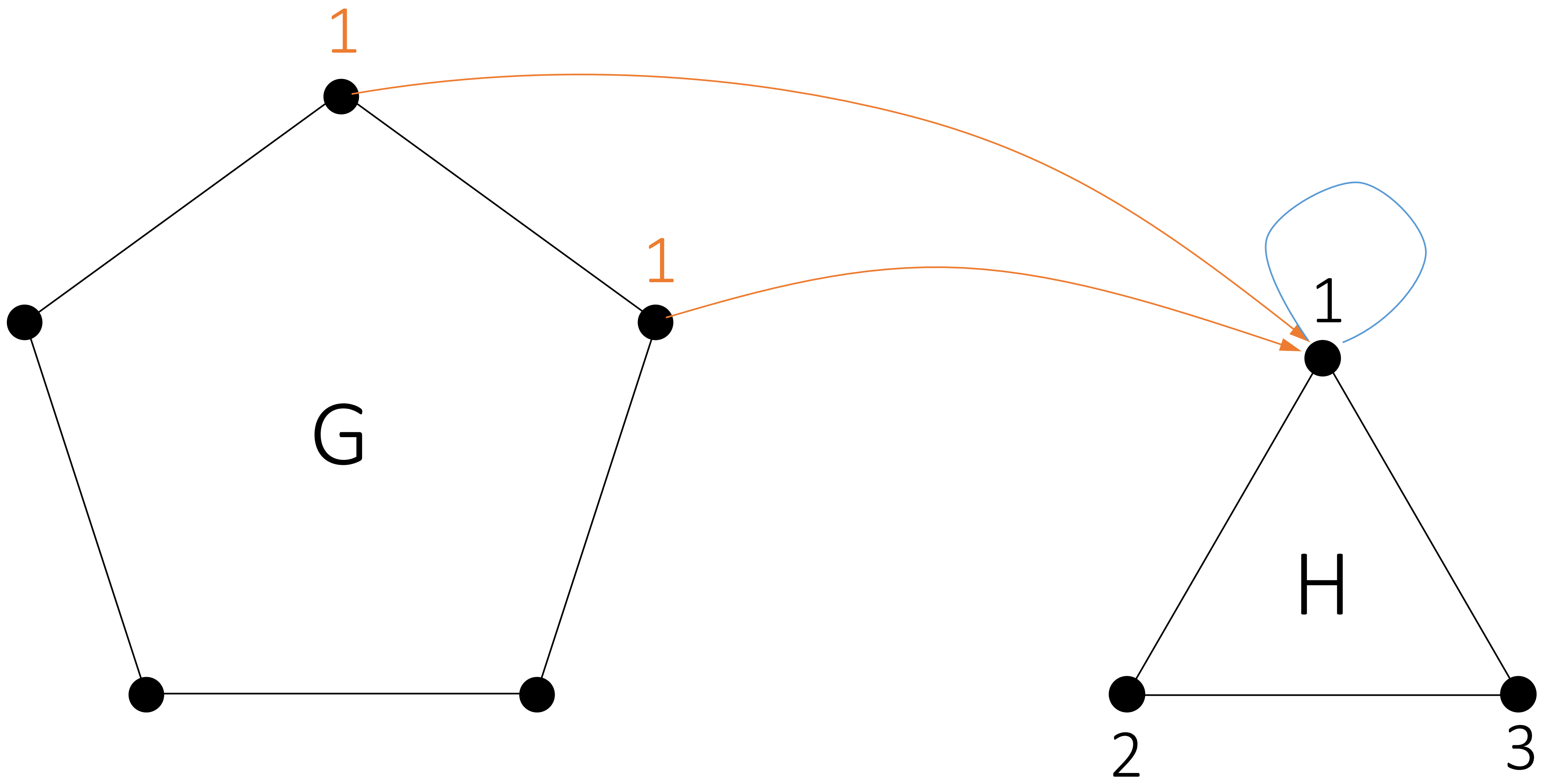
An example



Can this assignment be extended to a homomorphism?

Example by Rick Brewster, Graph homomorphism tutorial, 2006

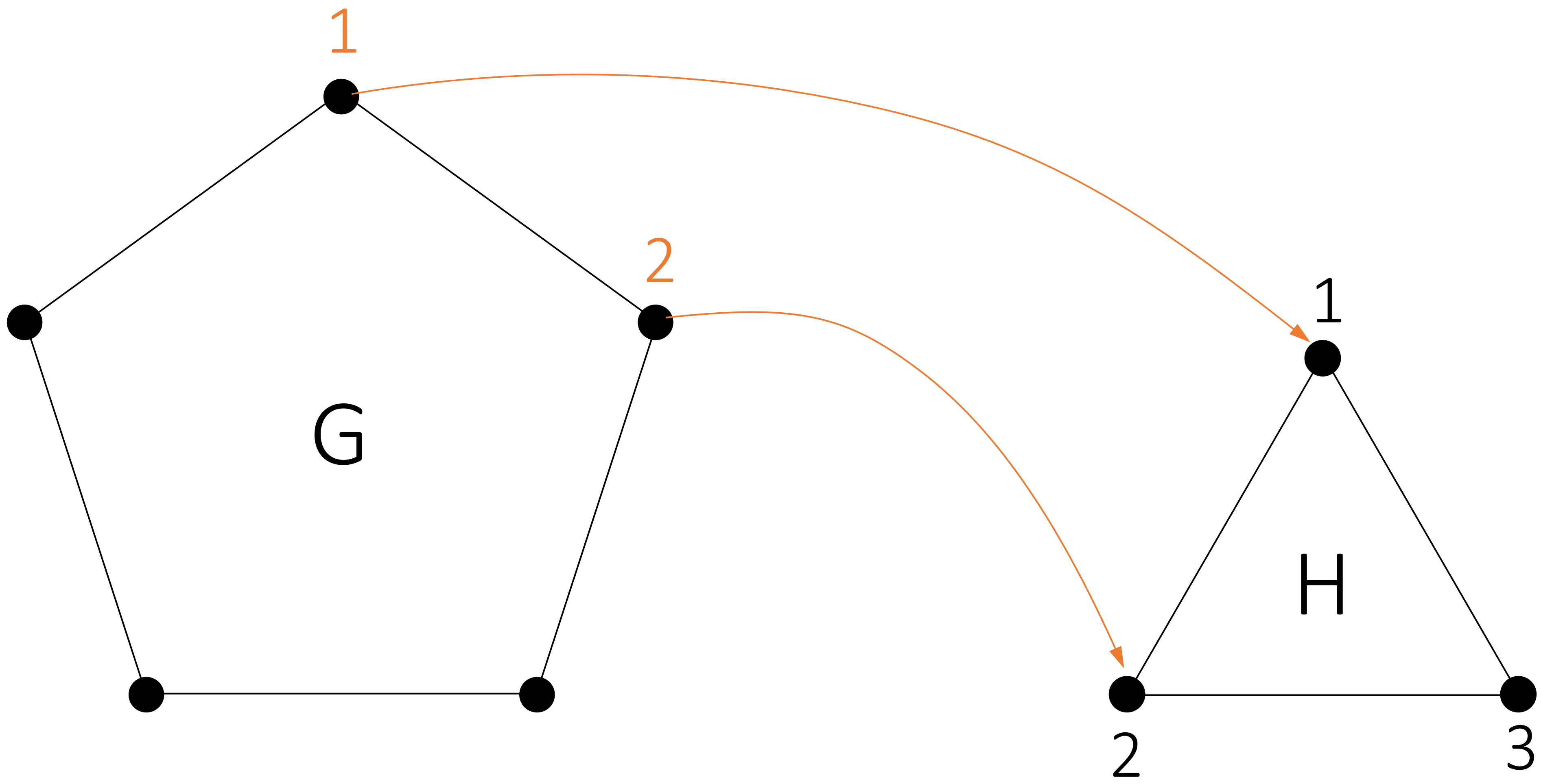
An example



No, this assignment requires a loop on vertex 1 (in H)

Example by Rick Brewster, Graph homomorphism tutorial, 2006

An example



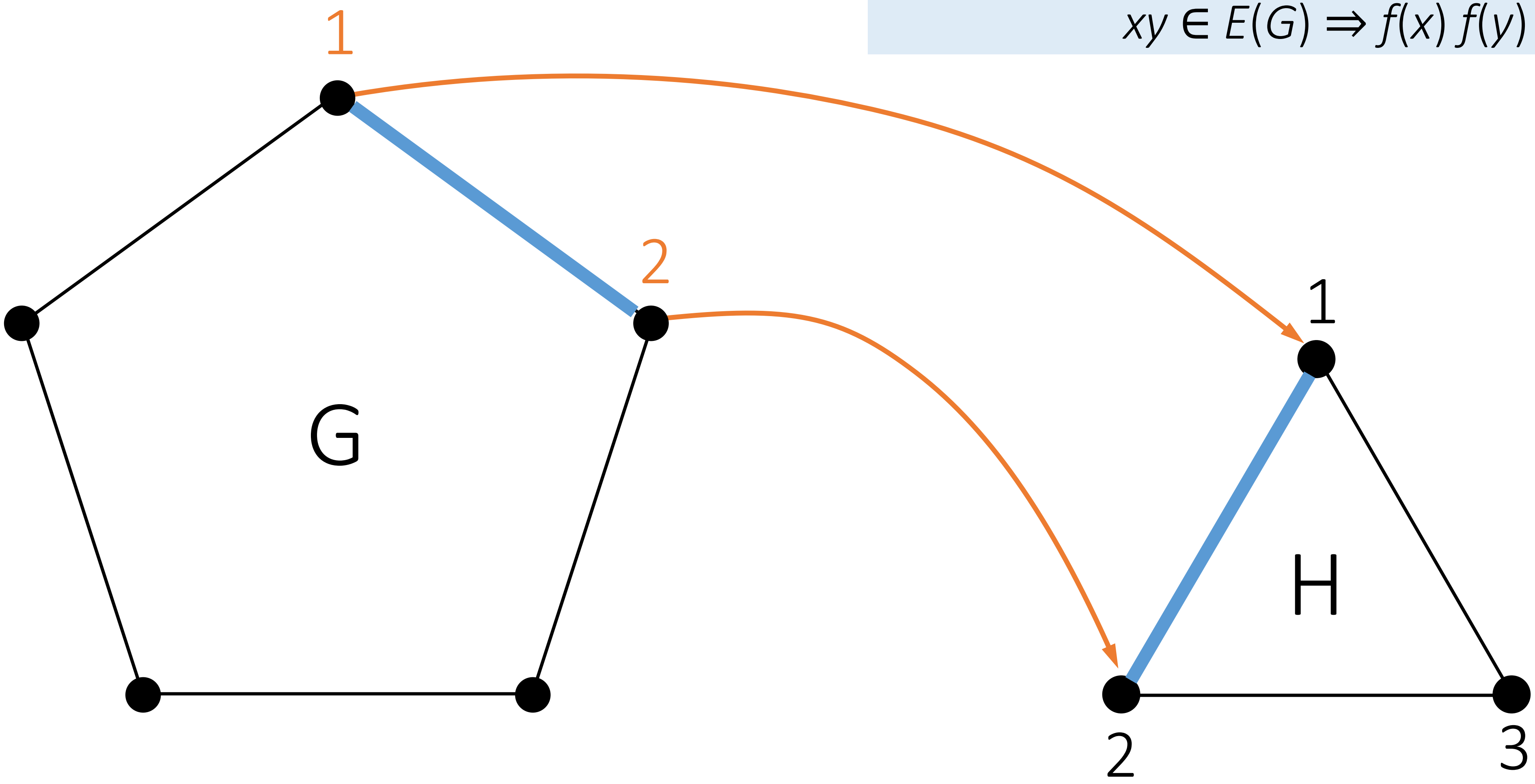
Is this assignment allowed?

Example by Rick Brewster, Graph homomorphism tutorial, 2006

An example

Definition: *Let G and H be graphs. A homom. of G to H is a function f*: $V(G) \to V(H)$ s.t. that

$$xy \in E(G) \Rightarrow f(x)\, f(y) \in E(H).$$



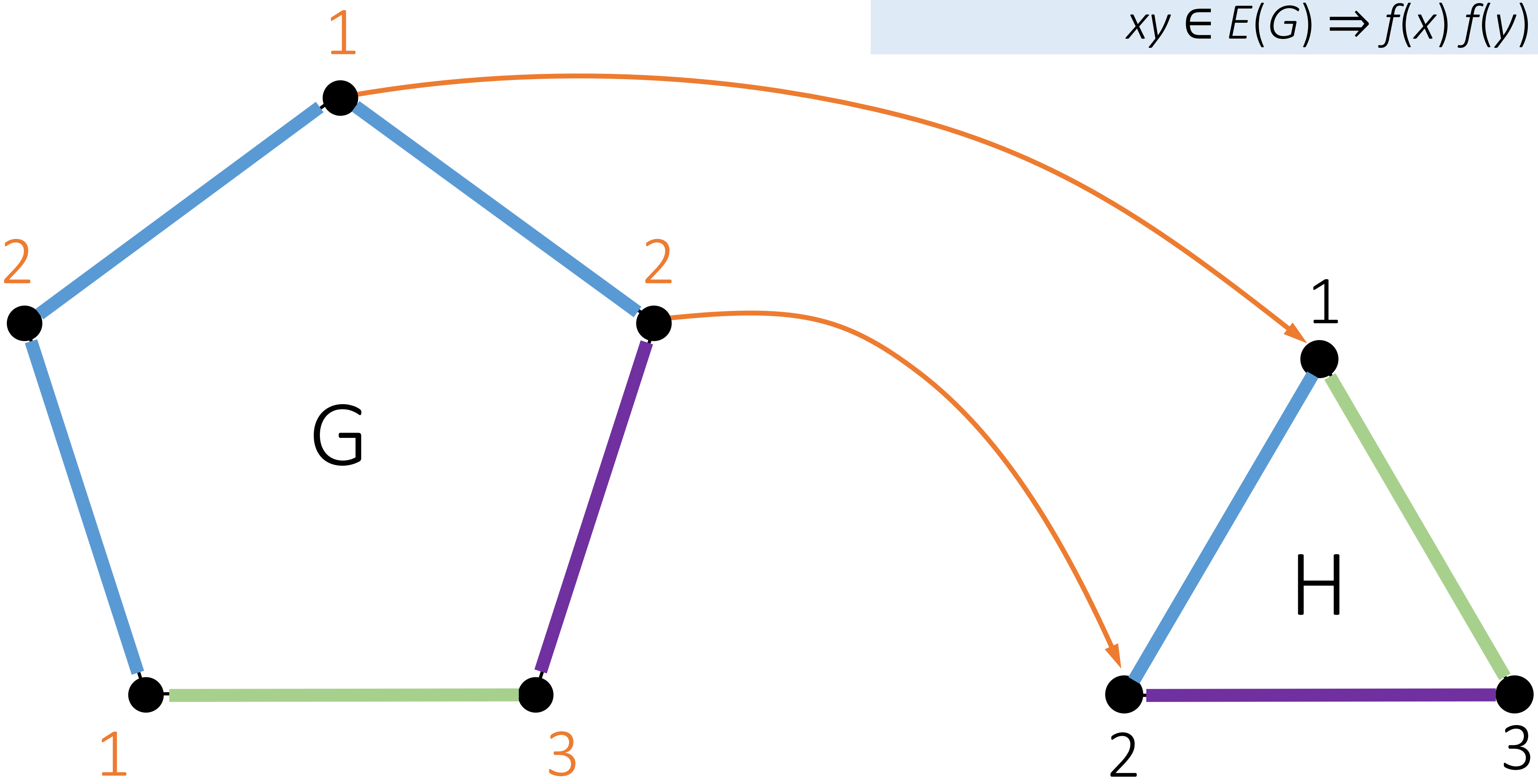
Is this assignment allowed?

Example by Rick Brewster, Graph homomorphism tutorial, 2006

An example

Definition: *Let G and H be graphs. A homom. of G to H is a function f*: $V(G) \to V(H)$ s.t. that

$$xy \in E(G) \Rightarrow f(x)\, f(y) \in E(H).$$

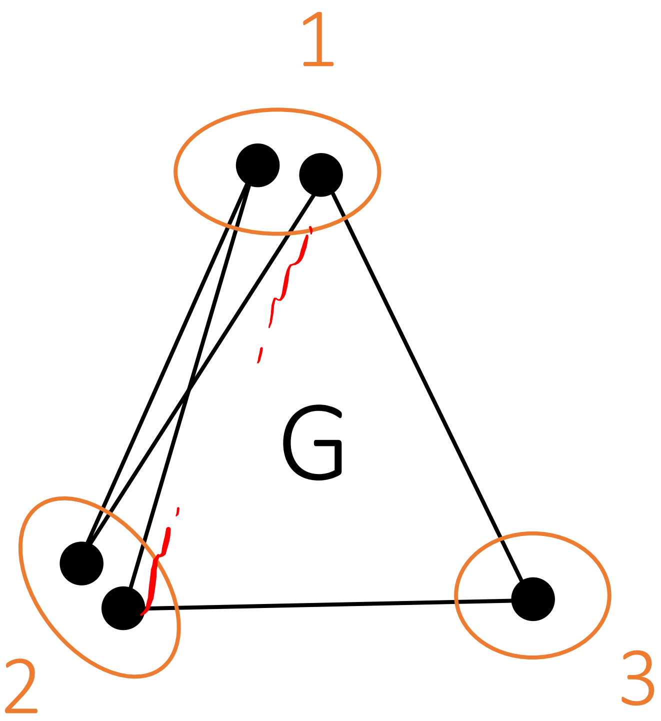


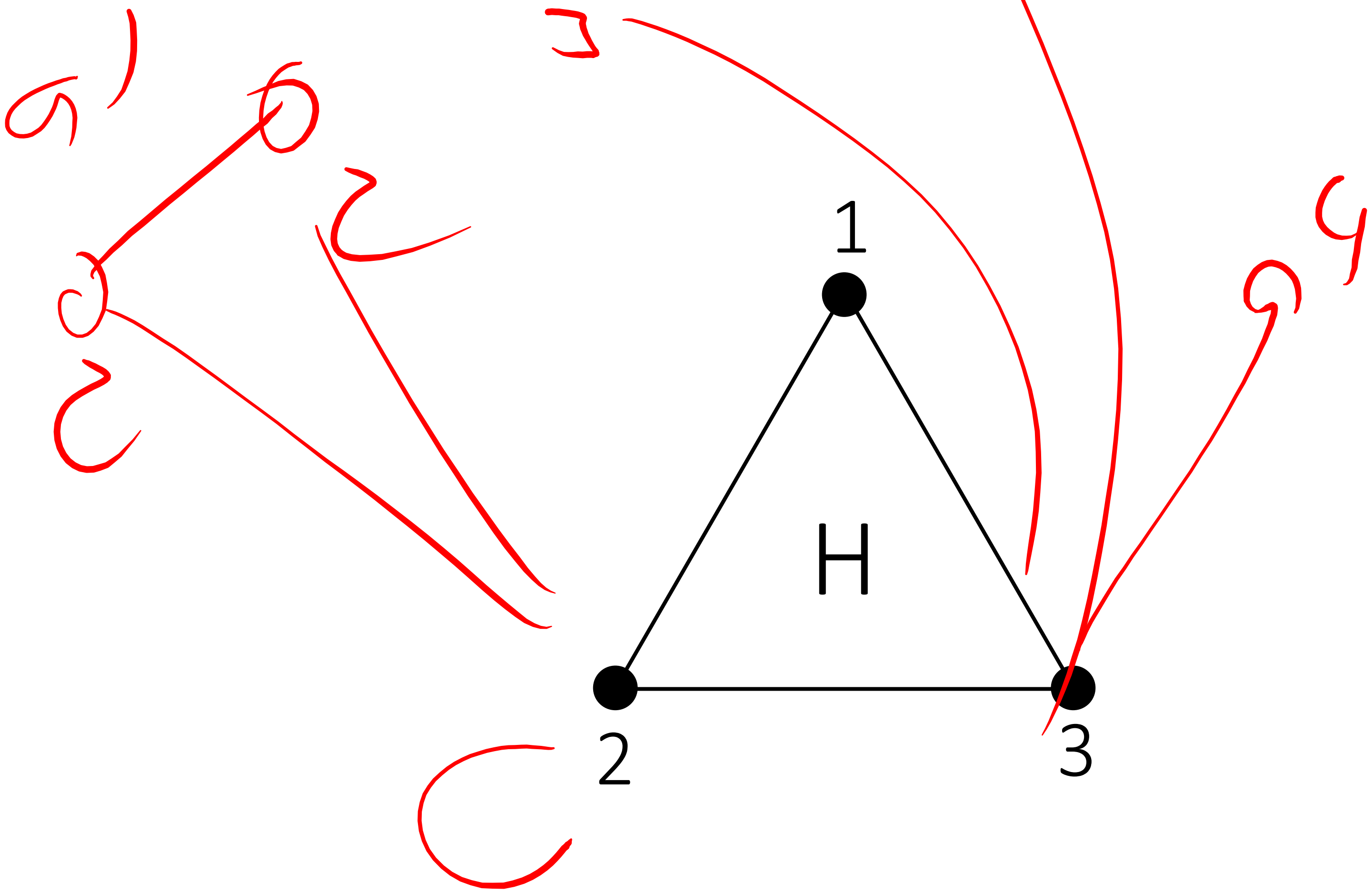
Example by Rick Brewster, Graph homomorphism tutorial, 2006

An example

Basically a partitioning problem!

The quotient of the partition (set of equivalences of the partition) is a subgraph of H.





Example by Rick Brewster, Graph homomorphism tutorial, 2006

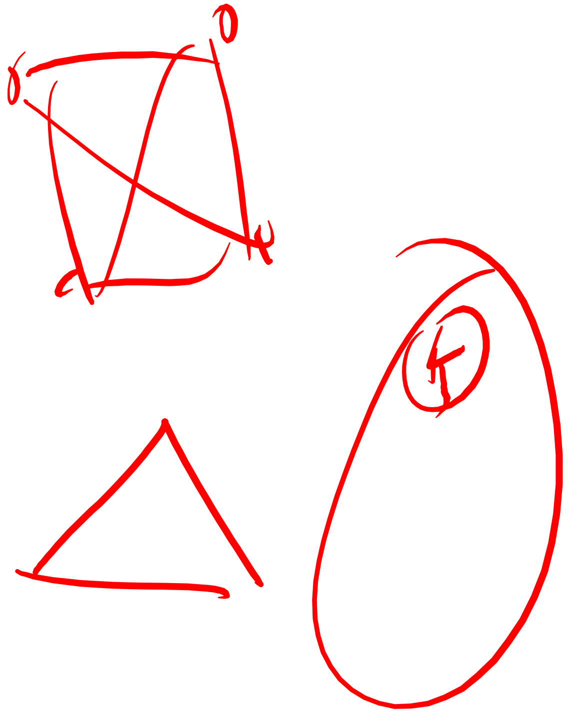
Some observations

When does $G \to K_3$ hold? (K_3 = 3-clique = triangle)

iff G is 3-colorable

When does $G \to K_n$ hold? (K_n = n-clique)

iff G is n-colorable



Thus homomorphisms generalize colorings:

Notation: $G \to H$ is an H-coloring of G.

What is the complexity of testing for the existence of a homomorphism?

NP-complete

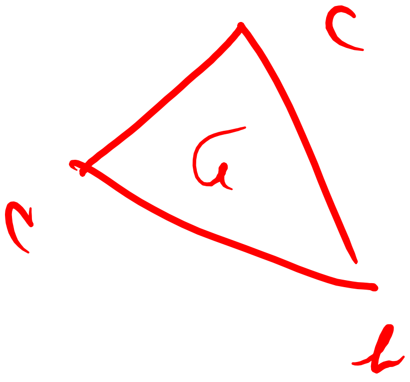
The complexity of H-coloring

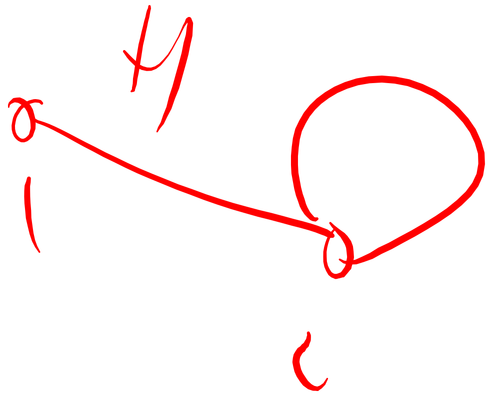
Let H be a fixed graph.

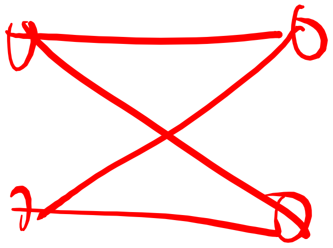
H-coloring

Instance: A graph G.

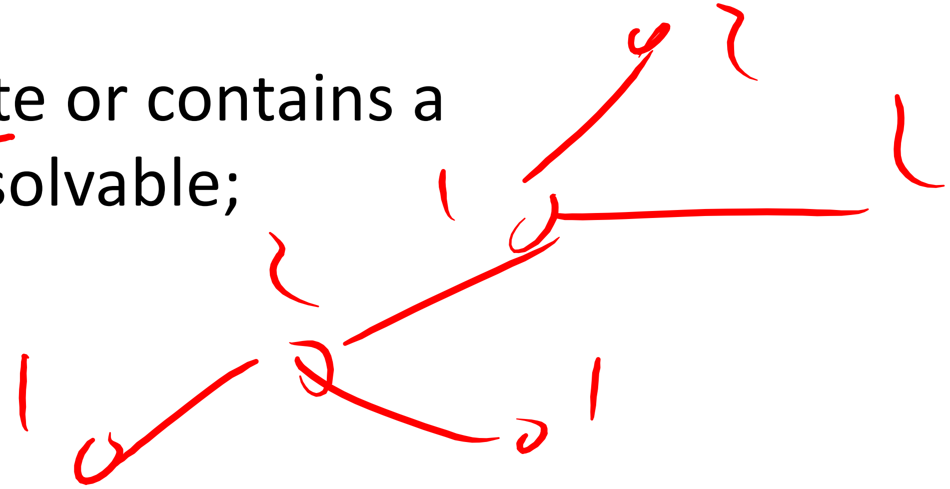
Question: Does G admit an H-coloring.







Theorem [Hell,Nesetril 1990] If H is bipartite or contains a loop, then H-colouring is polynomial time solvable; otherwise, H is NP-complete.



Repeated variable names

When evaluating a sentence with multiple quantifiers, don't fall into the trap of thinking that distinct variables range over distinct objects.

? Which of the following formulas imply each other?

$\forall x. \forall y.\ P(x,y)$

$\forall x.\ P(x,x)$

$\exists x. \exists y.\ P(x,y)$

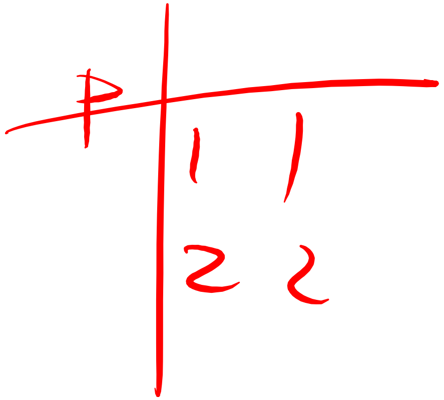
$\exists x.\ P(x,x)$

Repeated variable names

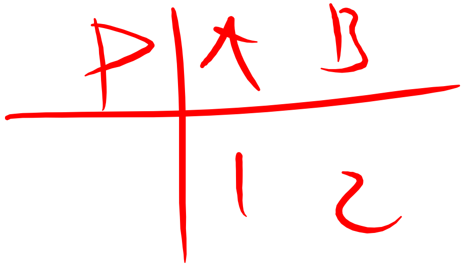
When evaluating a sentence with multiple quantifiers, don't fall into the trap of thinking that distinct variables range over distinct objects.

Recall that distinct variables do not need to range over distinct objects.

$\forall x. \forall y.\ P(x,y) \quad \Leftarrow \quad \forall x.\ P(x,x)$



$\exists x. \exists y.\ P(x,y) \quad \Rightarrow \quad \exists x.\ P(x,x)$



Repeated variable names

When evaluating a sentence with multiple quantifiers, don't fall into the trap of thinking that distinct variables range over distinct objects.

Recall that distinct variables do not need to range over distinct objects.

$$\forall x. \forall y.\ P(x,y) \quad \Leftarrow \quad \forall x.\ P(x,x)$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

Only if domain is not empty!
$Dom \neq \emptyset$

$$\exists x. \exists y.\ P(x,y) \quad \Rightarrow \quad \exists x.\ P(x,x)$$

Homomorphisms on Binary Structures

- **Definition (Binary algebraic structure):** A binary algebraic structure is a set together with a binary operation on it. This is denoted by an ordered pair $(S,\star)$ in which S is a set and $\star$ is a binary operation on S.

- **Definition (homomorphism of binary structures):** Let $(S,\star)$ and $(S',\circ)$ be binary structures. A homomorphism from $(S,\star)$ to $(S',\circ)$ is a map $h: S \longrightarrow S'$ that satisfies, for all x, y in S:

  $$h(x \star y) = h(x) \circ h(y)$$

- We can denote it by $h: (S,\star) \longrightarrow (S',\circ)$.

Examples

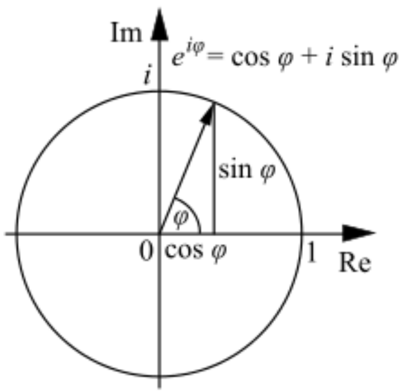
- Let $f(x) = e^x$. Then is f a homomorphism?
  - Yes, from the real numbers with addition $(\mathbb{R},+)$ to
  - the positive real numbers with multiplication $(\mathbb{R}^+,\cdot)$
  - even an isomorphism!

$f(x+y) = f(x) \cdot f(y)$

$f: (\mathbb{R},+) \longrightarrow (\mathbb{R}^+,\cdot)$

The exponential map $\exp : \mathbb{R} \to \mathbb{R}^+$ defined by $\exp(x) = e^x$, where e is the base of the natural logarithm, is an isomorphism from $(\mathbb{R}, +)$ to $(\mathbb{R}^+, \times)$. Exp is a bijection since it has an inverse function (namely $\log_e$) and exp preserves the group operations since $e^{x+y} = e^x e^y$. In this example both the elements and the operations are different yet the two groups are isomorphic, that is, as groups they have identical structures.

- Let $g(x) = e^{ix}$. Is g also a homomorphism?
  - Yes, from the real numbers with addition $(\mathbb{R},+)$ to
  - the unit circle in the complex plane with rotation



Paragraph screenshot from p.37 in 2004 - Dummit, Foote - Abstract algebra (book, 3rd ed).

Examples

$G = \mathbb{R}$ under $+$

$H = \{ z \in \mathbb{C} : |z| = 1 \}$

$= \text{Group under } \times$

Hint:

Every $z \in \mathbb{C}$ with $|z| = 1$ can be written as $z = e^{i\theta}$.

$f: G \to H$

$x \mapsto e^{ix}$

Show $f(x+y) = f(x) \times f(y)$

$$e^{i(x+y)} = e^{ix} \times e^{iy}$$

$$e^{ix+iy} = e^{ix} \times e^{iy}$$

$$e^{ix} \times e^{iy} = e^{ix} \times e^{iy}$$

$f(0) = f(2\pi) = 1, \quad f(2\pi n) = 1$

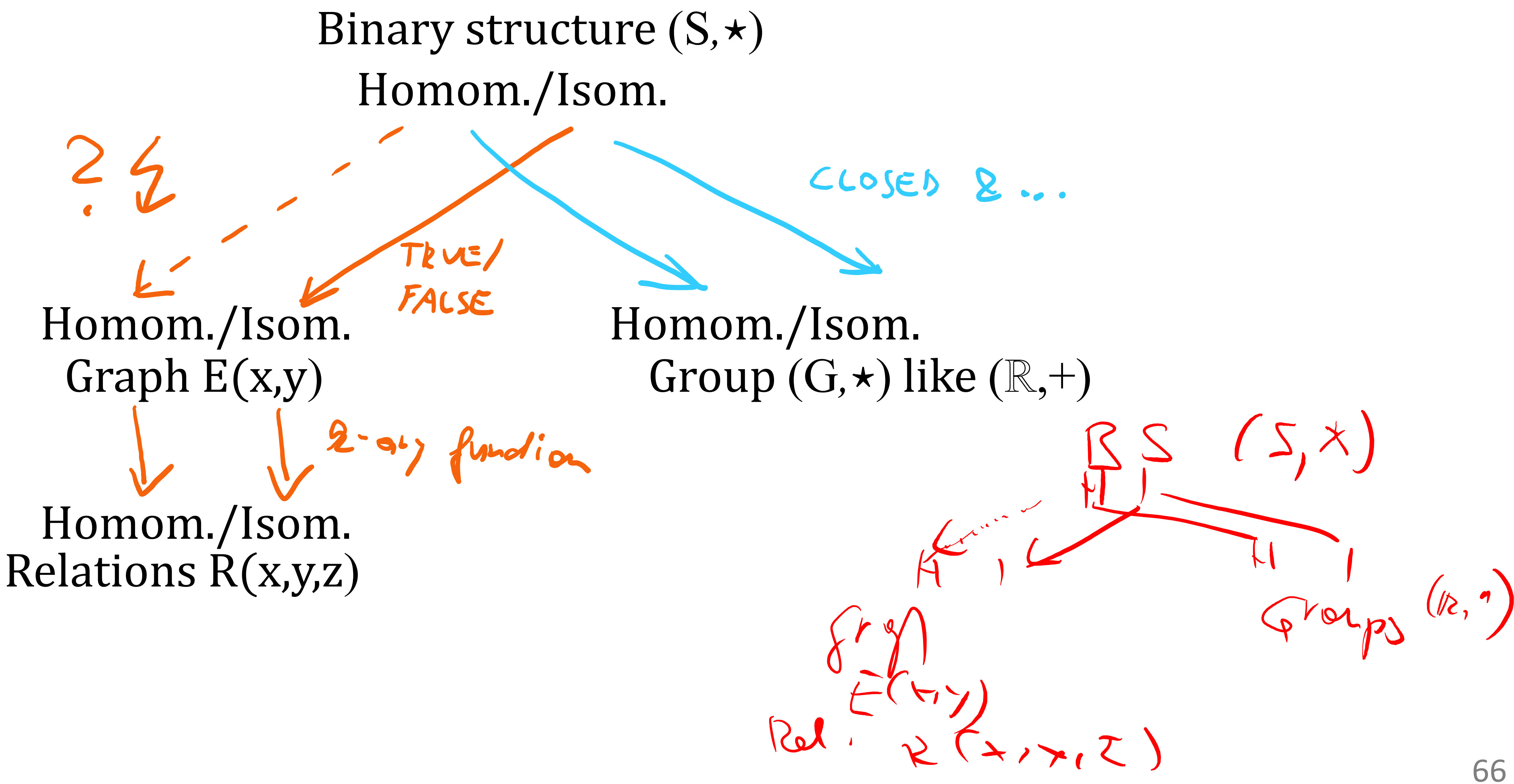
f is not 1-1

Also see: https://www.youtube.com/watch?v=cYzp5IWqCsg

Isomorphism

- **Definition**: A homomorphism of binary structures is called an isomorphism iff the corresponding map of sets is one to one and onto.

Some homomorphisms



Pointers to related work

- Kolaitis. *Logic and Databases*. Logical Structures in Computation Boot Camp, Berkeley 2016. https://simons.berkeley.edu/talks/logic-and-databases
- Abiteboul, Hull, Vianu. *Foundations of Databases*. Addison Wesley, 1995. http://webdam.inria.fr/Alice/, Ch 2.1: Theoretical background, Ch 6.2: Conjunctive queries & homomorphisms & query containment, Ch 6.3: Undecidability of equivalence for calculus.
- Kolaitis, Vardi. *Conjunctive-Query Containment and Constraint Satisfaction*. JCSS 2000. https://doi.org/10.1006/jcss.2000.1713
- Vardi. *Constraint satisfaction and database theory: a tutorial*. PODS 2000. https://doi.org/10.1145/335168.335209