



Logical foundations of databases

Diego Figueira

Gabriele Puppis

CNRS LaBRI



About the speakers...



Gabriele Puppis

PhD from Udine (Italy)

post-docs in Oxford

Works in LaBRI, Bordeaux

CNRS researcher



Diego Figueira

PhD from ENS Cachan (France),

post-docs in Warsaw, Edinburgh

Works in LaBRI, Bordeaux

CNRS researcher

First and foremost...



interrupt!

Organization

Schedule:

Relational databases

Relational Algebra

First-Order logic

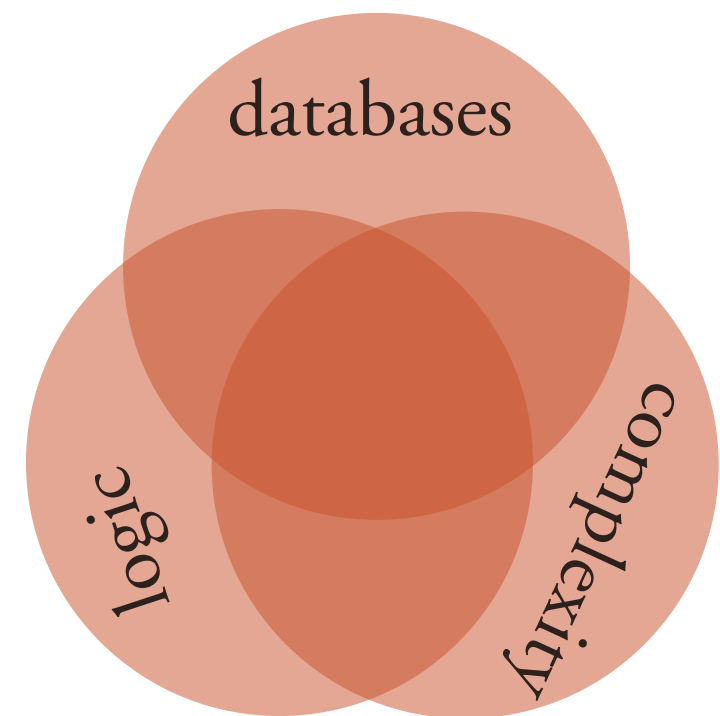
EF games

Locality

0-1 law

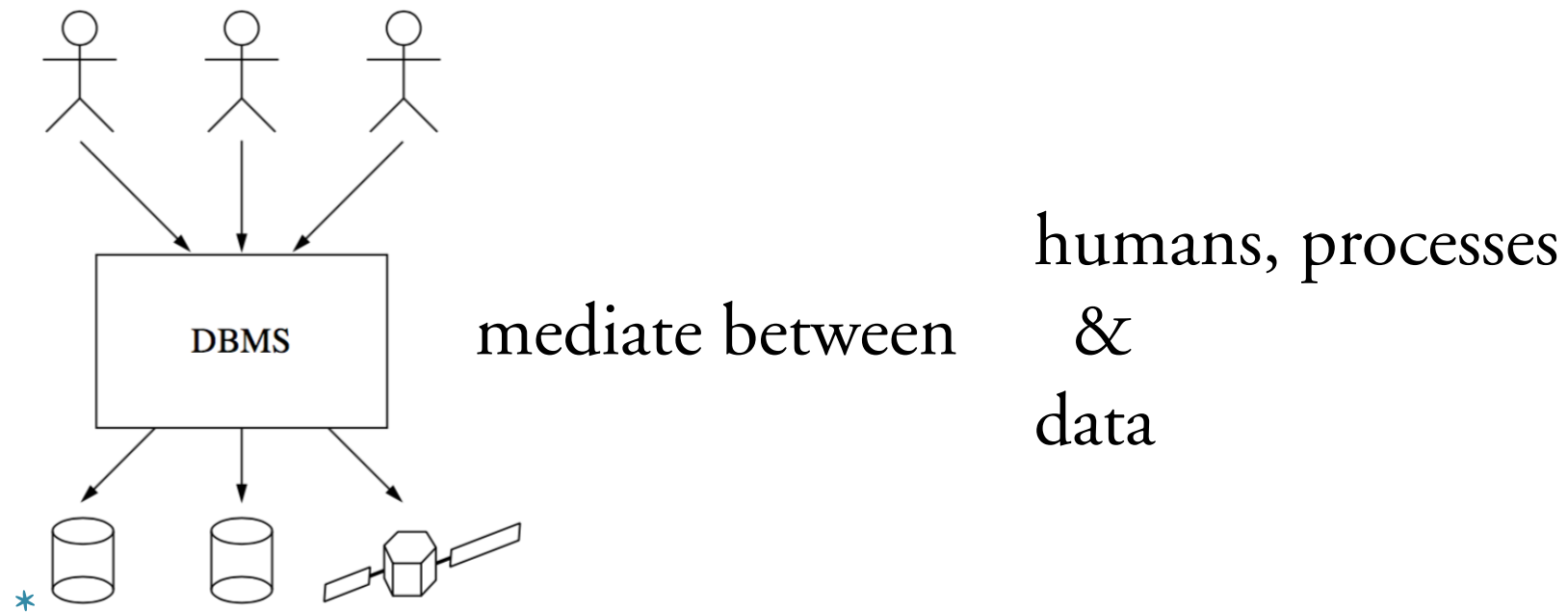
Conjunctive Queries

Acyclicity



Databases

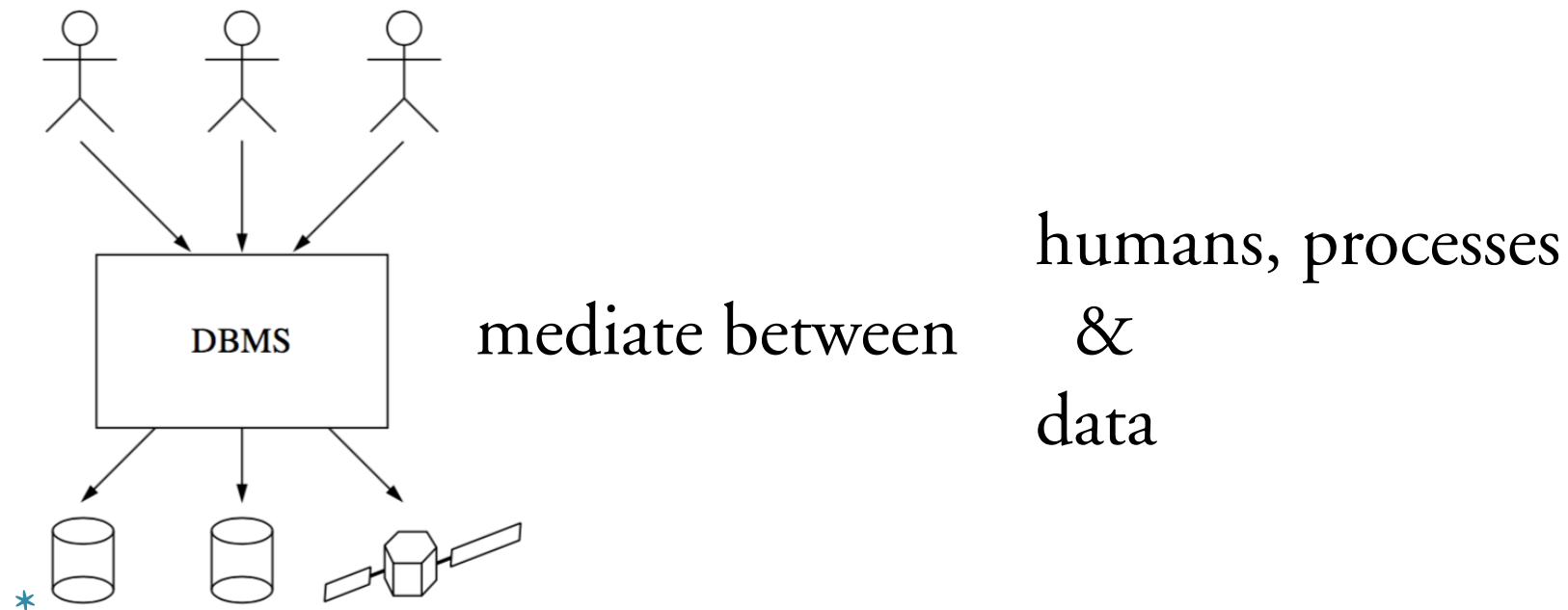
DBMS = a collection of data, structured in some way + a way of defining, querying, updating the data inside



* [Abitebou, Hull, Vianu “Foundations of databases”]

Databases

DBMS = a collection of data, structured in some way + a way of defining, querying, updating the data inside



Data model

- how the data is **logically organised**
- mathematical abstraction for representing data
- independent from physical organisation

DBMS also implement: transactions, concurrency, access control, resiliency...

* [Abitebou, Hull, Vianu "Foundations of databases"]

Relational databases, historical outlook

1970–72: E.F. Codd (IBM San Jose research lab) introduces the "relational data model" and two query languages: "relational algebra" and "relational calculus"

1974–75: IBM researchers start implementing

- "System R": first relational database management system (RDBMS).
- SEQUEL: a query language based on relational algebra

1983: IBM "DB2" is released, based on System R.

And UC Berkley released Ingres RDBMS

1979: Oracle Corporation is founded

1981: Codd receives Turing award

Now: multi-billion industry

Company	2006 Revenue	2006 Market Share
Oracle	7.168B	47.1%
IBM	3.204B	21.1%
Microsoft	2.654B	17.4%
Teradata	494.2M	3.2%
Sybase	486.7M	3.2%
Other	1.2B	7.8%
Total	15.2B	100%

Relational databases

Relational data model = data logically organised into relations (“tables”).

What's a **relation**?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

Relational databases

Relational data model = data logically organised into relations (“tables”).

What's a **relation**?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

like:

$$\{ (1,a,2), (2,b,6), (2,a,1) \} \subseteq \mathbb{N} \times \{a,b\} \times \mathbb{N}$$

Relational databases

Relational data model = data logically organised into relations (“tables”).

What’s a **relation**?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

like:

$\{ (1,a,2), (2,b,6), (2,a,1) \} \subseteq \mathbb{N} \times \{a,b\} \times \mathbb{N}$

→ a “tuple” (a “3-tuple”)

Relational databases

Relational data model = data logically organised into relations (“tables”).

What's a **relation**?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

like:

$$\{ (1,a,2), (2,b,6), (2,a,1) \} \subseteq \mathbb{N} \times \{a,b\} \times \mathbb{N}$$

→ a “tuple” (a “3-tuple”)

() → 0-tuple

Relational databases

Relational data model = data logically organised into relations (“tables”).

What’s a **relation**?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

like:

$\{ (1,a,2), (2,b,6), (2,a,1) \} \subseteq \mathbb{N} \times \{a,b\} \times \mathbb{N}$

→ a “tuple” (a “3-tuple”)

$() \rightarrow$ 0-tuple

like: “

1	a	2
2	b	6
2	a	1

”

Relational databases

Relational data model = data logically organised into relations (“tables”).

What’s a **relation**?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

DB = A **schema**: names of tables and attributes

An **instance**: data conforming to the schema

Relational databases

Relational data model = data logically organised into relations (“tables”).

What’s a **relation**?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

DB = A **schema**: names of tables and attributes

Films (Title:string, Director:string, Actor:string)

Schedule (Theatre:string, Title:string)

An **instance**: data conforming to the schema

Relational databases

Relational data model = data logically organised into relations (“tables”).

What’s a **relation**?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

DB = A **schema**: names of tables and attributes

Films (Title:string, Director:string, Actor:string)

Schedule (Theatre:string, Title:string)

An **instance**: data conforming to the schema

Films

Title	Director	Actor
8 1/2	Fellini	Mastroianni
Shining	Kubrick	Nicholson
Dr. Strangelove	Kubrick	Sellers
8 femmes	Ozon	Ardant

Schedule

Theatre	Title
Utopia	Dr. Strangelove
Utopia	8 1/2
UGC	Dr. Strangelove
UGC	8 femmes

Relational databases

Relational data model = data logically organised into relations (“tables”).



We assume all elements come from
a fixed set of *constants* or *data values* U .

Relational databases: queries

- What is a query q ?
 - A mapping that takes a database instance D and returns a relation $q(D) \subseteq U^r$ of fixed arity r

Relational databases: queries

computable!

What is a query q ?

A mapping that takes a database instance D
returns a relation $q(D) \subseteq U^r$ of fixed arity r

Relational databases: queries

computable!

What is a query q ?

A mapping that takes a database instance D
returns a relation $q(D) \subseteq U^r$ of fixed arity r

generic!
(order independent)

Relational databases: queries

computable!

What is a query q ?

A mapping that takes a database instance D
returns a relation $q(D) \subseteq U^r$ of fixed arity r

generic!
(order independent)

Boolean query: $r=0$
Either “yes” $\{ () \}$ or “no” $\{ \}$

Relational databases: queries

What is a query q ?

A mapping that takes a database instance D
returns a relation $q(D) \subseteq U^r$ of fixed arity r

What do we care about queries?



expressive power



evaluation



static analysis

The fundamental questions:

How to query the relational data model?

How efficient/expressive is it?

The fundamental questions:

How to query the relational data model?

How efficient/expressive is it?



Query Language

Syntax

+

Semantics

Expressions for querying the db,
governed by syntactic rules

Interpretation of symbols
in terms of some structure

“Select X from Y”

Retrieves all strings
in column X of table Y

“ $y :- \forall x (x \leq y)$ ”

Returns the maximum element
of the set.

Syntax: $E := R, S, \dots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E)$

where $M \subseteq \mathbb{N}$

$\Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N}$

Syntax: $E := R, S, \dots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E)$

where $M \subseteq \mathbb{N}$

$\Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N}$

- $R_1 \cup R_2$: Set union
- $R_1 \times R_2$: Cartesian product
- $R_1 \setminus R_2$: Set difference

Syntax: $E := R, S, \dots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E)$

where $M \subseteq \mathbb{N}$

$\Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N}$

- $R_1 \cup R_2$: Set union
- $R_1 \times R_2$: Cartesian product
- $R_1 \setminus R_2$: Set difference
- $\sigma_{\{i_1=j_1, \dots, i_n=j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1}=x_{j_1}) \wedge \dots \wedge (x_{i_n}=x_{j_n})\}$: Selection

Syntax: $E := R, S, \dots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E)$

where $M \subseteq \mathbb{N}$

$\Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N}$

- $R_1 \cup R_2$: Set union
- $R_1 \times R_2$: Cartesian product
- $R_1 \setminus R_2$: Set difference
- $\sigma_{\{i_1 \neq j_1, \dots, i_n \neq j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$: Selection

Syntax: $E := R, S, \dots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E)$

where $M \subseteq \mathbb{N}$

$\Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N}$

- $R_1 \cup R_2$: Set union
- $R_1 \times R_2$: Cartesian product
- $R_1 \setminus R_2$: Set difference

$$\sigma_{\{1=3, 1 \neq 2\}}(\{(1, 2, 1), (2, 2, 2)\}) = \{(1, 2, 1)\}$$

- $\sigma_{\{i_1 \neq j_1, \dots, i_n \neq j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$: Selection

Syntax: $E := R, S, \dots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E)$

where $M \subseteq \mathbb{N}$

$\Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N}$

- $R_1 \cup R_2$: Set union

- $R_1 \times R_2$: Cartesian product

- $R_1 \setminus R_2$: Set difference

- $\sigma_{\{i_1 \neq j_1, \dots, i_n \neq j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$: Selection

- $\pi_{\{i_1, \dots, i_n\}}(R) := \{(x_{i_1}, \dots, x_{i_n}) \mid (x_1, \dots, x_m) \in R\}$: Projection

$$\sigma_{\{1=3, 1 \neq 2\}}(\{(1, 2, 1), (2, 2, 2)\}) = \{(1, 2, 1)\}$$

Syntax: $E := R, S, \dots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E)$

where $M \subseteq \mathbb{N}$

$\Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N}$

- $R_1 \cup R_2$: Set union

- $R_1 \times R_2$: Cartesian product

- $R_1 \setminus R_2$: Set difference

- $\sigma_{\{i_1 \neq j_1, \dots, i_n \neq j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$: Selection

- $\pi_{\{i_1, \dots, i_n\}}(R) := \{(x_{i_1}, \dots, x_{i_n}) \mid (x_1, \dots, x_m) \in R\}$: Projection

$$\sigma_{\{1=3, 1 \neq 2\}}(\{(1, 2, 1), (2, 2, 2)\}) = \{(1, 2, 1)\}$$

$$\pi_{\{1, 3\}}(\{(1, 2, 1), (2, 2, 2)\}) = \{(1, 1), (2, 2)\}$$

- $R_1 \cup R_2$: Set union
- $R_1 \times R_2$: Cartesian product
- $R_1 \setminus R_2$: Set difference
- $\sigma_{\{i_1=j_1, \dots, i_n=j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$: Selection
- $\pi_{\{i_1, \dots, i_n\}}(R) := \{(x_{i_1}, \dots, x_{i_n}) \mid (x_1, \dots, x_m) \in R\}$: Projection

Question 1: What is the RA expression for

$\{ (v_1, v_2) \mid \text{there are } w_1 \neq w_2 \text{ so that } (v_1, w_1) \in R_1 \text{ and } (v_2, w_2) \in R_2 \}$?

Question 2: $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2)) = ?$

R_1		R_2	
a	3	a	4
b	2	b	1
c	4	b	2
b	3	a	1
a	2	b	3

- $R_1 \cup R_2$: Set union
- $R_1 \times R_2$: Cartesian product
- $R_1 \setminus R_2$: Set difference
- $\sigma_{\{i_1 \neq j_1, \dots, i_n \neq j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$: Selection
- $\pi_{\{i_1, \dots, i_n\}}(R) := \{(x_{i_1}, \dots, x_{i_n}) \mid (x_1, \dots, x_m) \in R\}$: Projection

Question 1: What is the RA expression for

$\{(v_1, v_2) \mid \text{there are } w_1 \neq w_2 \text{ so that } (v_1, w_1) \in R_1 \text{ and } (v_2, w_2) \in R_2\}$?

Answer: $\pi_{\{1,3\}}(\sigma_{1 \neq 3}(R_1 \times R_2))$

a	b
b	a
c	a
c	b

Question 2: $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2)) = ?$

R_1		R_2	
a	3	a	4
b	2	b	1
c	4	b	2
b	3	a	1
a	2	b	3

- $R_1 \cup R_2$: Set union
- $R_1 \times R_2$: Cartesian product
- $R_1 \setminus R_2$: Set difference
- $\sigma_{\{i_1=j_1, \dots, i_n=j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$: Selection
- $\pi_{\{i_1, \dots, i_n\}}(R) := \{(x_{i_1}, \dots, x_{i_n}) \mid (x_1, \dots, x_m) \in R\}$: Projection

Question 1: What is the RA expression for

$\{(v_1, v_2) \mid \text{there are } w_1 \neq w_2 \text{ so that } (v_1, w_1) \in R_1 \text{ and } (v_2, w_2) \in R_2\}$?

Answer: $\pi_{\{1,3\}}(\sigma_{1 \neq 3}(R_1 \times R_2))$

a	b
b	a
c	a
c	b

Question 2: $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2)) = ?$

Answer (only one element):

b

R_1		R_2	
a	3	a	4
b	2	b	1
c	4	b	2
b	3	a	1
a	2	b	3

RA = Basic SQL

no domain-specific features,
aggregation, etc

Select X
From R_1, \dots, R_n $\iff \pi_X (\sigma_Z(R_1 \times \dots \times R_n))$
Where Z

... or ... \iff union
... not in (...) \iff difference

RA = Basic SQL

no domain-specific features,
aggregation, etc

Select X
From R_1, \dots, R_n $\iff \pi_X (\sigma_Z (R_1 \times \dots \times R_n))$
Where Z

... or ... \iff union
... not in (...) \iff difference

$\pi_2 (\sigma_{1 \neq 3} (R_1 \times R_2)) \rightsquigarrow$

R_1		R_2	
a	3	a	4
b	2	b	1
c	4	b	2
b	3	a	1
a	2	b	3

RA = Basic SQL

no domain-specific features,
aggregation, etc

Select X
From R_1, \dots, R_n $\iff \pi_X (\sigma_Z (R_1 \times \dots \times R_n))$
Where Z

... or ... \iff union
... not in (...) \iff difference

$\pi_2 (\sigma_{1 \neq 3} (R_1 \times R_2)) \rightsquigarrow$ Select $R_1.2$ as foo
From R_1, R_2
Where $R_1.1 \neq R_2.1$

R_1		R_2	
a	3	a	4
b	2	b	1
c	4	b	2
b	3	a	1
a	2	b	3

RA = Basic SQL

no domain-specific features,
aggregation, etc

Select X
From R_1, \dots, R_n $\iff \pi_X (\sigma_Z (R_1 \times \dots \times R_n))$
Where Z

... or ... \iff union
... not in (...) \iff difference

$\pi_2 (\sigma_{1 \neq 3} (R_1 \times R_2))$

\ast

Select $R_1.2$ as foo
From R_1, R_2
Where $R_1.1 \neq R_2.1$

R_1

a	3
b	2
c	4
b	3
a	2

R_2

a	4
b	1
b	2
a	1
b	3

$\pi_2 (\sigma_{1=3} (\ast \times R_2)) \rightsquigarrow$

RA = Basic SQL

no domain-specific features,
aggregation, etc

Select X
From R_1, \dots, R_n $\iff \pi_X (\sigma_Z (R_1 \times \dots \times R_n))$
Where Z

... or ... \iff union
... not in (...) \iff difference

$\pi_2 (\sigma_{1 \neq 3} (R_1 \times R_2)) \rightsquigarrow$
 $\underbrace{\hspace{10em}}_{*}$
 $\left(\begin{array}{l} \text{Select } R_1.2 \text{ as } \underline{\text{foo}} \\ \text{From } R_1, R_2 \\ \text{Where } R_1.1 \neq R_2.1 \end{array} \right) \star$

$\pi_2 (\sigma_{1=3} (* \times R_2)) \rightsquigarrow$

R_1		R_2	
a	3	a	4
b	2	b	1
c	4	b	2
b	3	a	1
a	2	b	3

RA = Basic SQL

no domain-specific features,
aggregation, etc

Select X
From R_1, \dots, R_n $\iff \pi_X (\sigma_Z (R_1 \times \dots \times R_n))$
Where Z

... or ... \iff union
... not in (...) \iff difference

$\pi_2 (\sigma_{1 \neq 3} (R_1 \times R_2))$

\Join

$\pi_2 (\sigma_{1=3} (\Join \times R_2))$

Select $R_1.2$ as foo
From R_1, R_2
Where $R_1.1 \neq R_2.1$

★

Select foo
From ★, R_2
Where $foo = R_2.2$

R_1

a	3
b	2
c	4
b	3
a	2

R_2

a	4
b	1
b	2
a	1
b	3

Denotational languages

Procedural

Algebra \leadsto *How* to obtain the result

Logics \leadsto *What* is the property of the result

Declarative

Denotational languages

  **Relational Algebra**
operations on tables

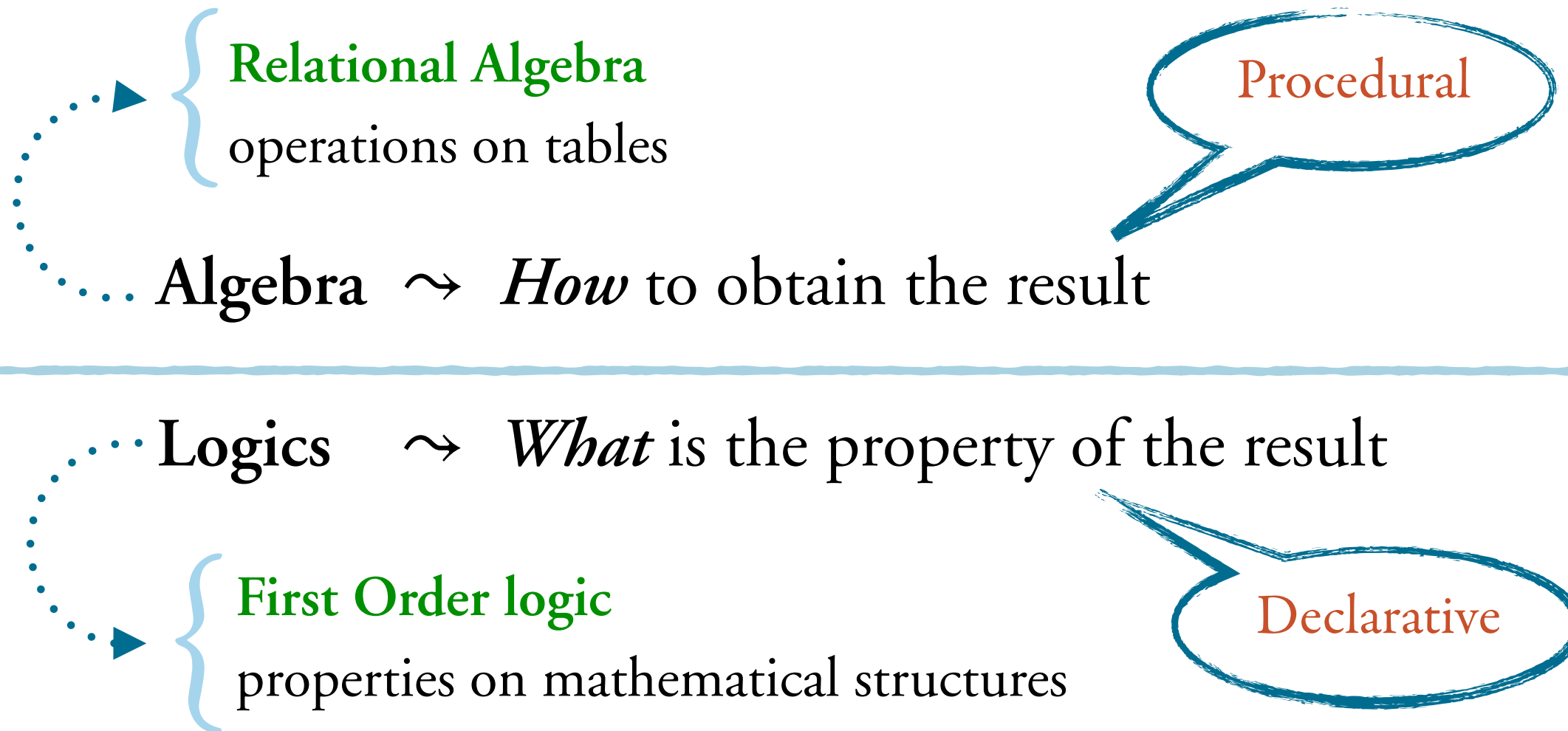
... Algebra \leadsto *How* to obtain the result

Procedural

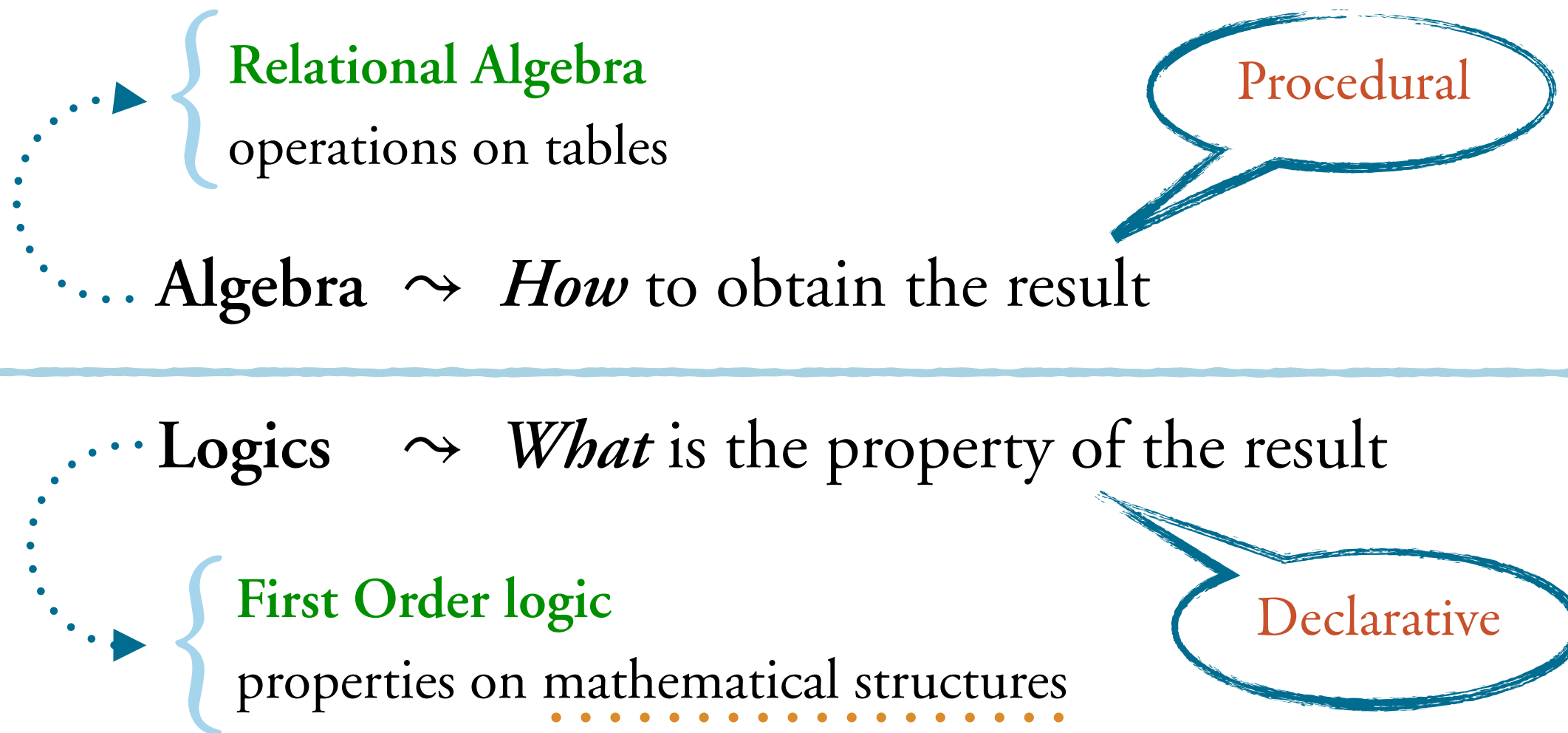
Logics \leadsto *What* is the property of the result

Declarative

Denotational languages



Denotational languages



FO = First-Order logic



Relational structures

A **structure** is:

$$A = (D, R_1, \dots, R_n, f_1, \dots, f_n)$$

D is a non-empty set, the domain

R_i is an m -ary relation for some m (ie, $R_i \subseteq D^m$)

f_i is an n -ary function for some n (ie, $f_i: D^n \longrightarrow D$)

Relational structures

A **structure** is:

$$A = (D, R_1, \dots, R_n, f_1, \dots, f_n)$$

D is a non-empty set, the domain

R_i is an m -ary relation for some m (ie, $R_i \subseteq D^m$)

f_i is an n -ary function for some n (ie, $f_i: D^n \longrightarrow D$)

A **graph** $G = (V, E)$

- V : nodes
- $E \subseteq V^2$: edges (binary relation)
- (no functions)

A **group**, like $(\mathbb{N}, +)$

- \mathbb{N} : natural numbers
- (no relations)
- $+: \mathbb{N}^2 \longrightarrow \mathbb{N}$ addition (binary function)

First-order logic

First-order logic

FO *variables* x, y, z, \dots
quantifiers: \exists, \forall
Boolean connectives: \neg, \wedge, \vee

A language to talk about **structures**

Variables range over the **domain**

Atomic formulas: $R(x_1, \dots, x_m), x=y$

First-order logic

FO *variables* x, y, z, \dots
quantifiers: \exists, \forall
Boolean connectives: \neg, \wedge, \vee

A language to talk about **structures**

Variables range over the **domain**

Atomic formulas: $R(x_1, \dots, x_m), x=y$

A **graph** $G = (V, E)$

- V : nodes
- $E \subseteq V^2$: edges (binary relation)
- (no functions)

Language to talk about **graphs**

Variables range over **nodes**

Atomic formulas: $E(x, y), x = y$

First-order logic

FO *variables* x, y, z, \dots
quantifiers: \exists, \forall
Boolean connectives: \neg, \wedge, \vee

A language to talk about **structures**

Variables range over the **domain**

Atomic formulas: $R(x_1, \dots, x_m), x=y$

A **graph** $G = (V, E)$

- V : nodes
- $E \subseteq V^2$: edges (binary relation)
- (no functions)

Language to talk about **graphs**

Variables range over **nodes**

Atomic formulas: $E(x, y), x = y$


Formulas: Atomic formulas + connectives + quantifiers

“The node x has at least two neighbours”

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

“The node x has at least two neighbours”


$$\varphi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

 free

x is **free** = not quantified
(a property of a node in the graph)

“The node x has at least two neighbours”

$$\varphi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

 free


x is **free** = not quantified
(a property of a node in the graph)

“Each node has at least two neighbours”

$$\forall x \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

“The node x has at least two neighbours”

$$\varphi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

 **free**

x is **free** = not quantified
(a property of a node in the graph)


“Each node has at least two neighbours”

$$\psi = \forall x \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

the formula is a **sentence**
= no free variables
(a property of the graph)

“The node x has at least two neighbours”

$$\varphi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

 **free**

x is **free** = not quantified
(a property of a node in the graph)

“Each node has at least two neighbours”

$$\psi = \forall x \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

the formula is a **sentence**
= no free variables
(a property of the graph)


Question: • How to express in FO

“Every two adjacent nodes have a common neighbour” ?

- Does it have free variables? Is it a sentence?

“The node x has at least two neighbours”

$$\varphi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

 **free**

x is **free** = not quantified
(a property of a node in the graph)

“Each node has at least two neighbours”

$$\psi = \forall x \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

the formula is a **sentence**
= no free variables
(a property of the graph)

Question: • How to express in FO

“Every two adjacent nodes have a common neighbour” ?

• Does it have free variables? Is it a sentence?

Answer: $\forall x \forall y \left(\neg E(x,y) \vee \exists z \left(\left(E(x,z) \vee E(z,x) \right) \wedge \left(E(y,z) \vee E(z,y) \right) \right) \right)$

Binding

To evaluate a formula ϕ we need a graph $G=(V,E)$ and a **binding** α that maps free variables of ϕ to nodes of G .

$G \models_{\alpha} \phi(x_1, \dots, x_n)$ $\alpha : \{x_1, \dots, x_n\} \longrightarrow V$ assigns nodes to free variables

Binding

To evaluate a formula ϕ we need a graph $G=(V,E)$ and a **binding** α that maps free variables of ϕ to nodes of G .

$\dots \blacktriangleright$ “ G, α satisfy ϕ ” $\dots \blacktriangleright$ “ ϕ is satisfiable”

$G \models_{\alpha} \phi(x_1, \dots, x_n)$ $\alpha : \{x_1, \dots, x_n\} \longrightarrow V$ assigns nodes to free variables

Binding

To evaluate a formula ϕ we need a graph $G=(V,E)$ and a **binding** α that maps free variables of ϕ to nodes of G .

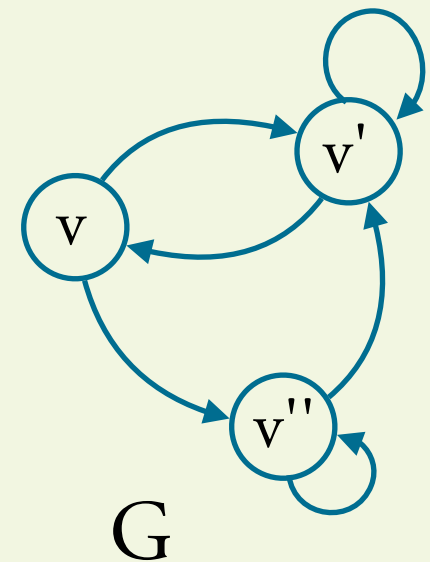
.....► “ G, α satisfy ϕ ”► “ ϕ is satisfiable”

$G \models_{\alpha} \phi(x_1, \dots, x_n)$ $\alpha : \{x_1, \dots, x_n\} \longrightarrow V$ assigns nodes to free variables

“*The node x has at least two neighbours*”

$$\phi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

$$G \models_{\alpha} \phi \quad \text{if} \quad \alpha = \{x \mapsto v\}$$



Binding

To evaluate a formula ϕ we need a graph $G=(V,E)$ and a **binding** α that maps free variables of ϕ to nodes of G .

.....► “ G, α satisfy ϕ ”► “ ϕ is satisfiable”

$G \models_{\alpha} \phi(x_1, \dots, x_n)$ $\alpha : \{x_1, \dots, x_n\} \longrightarrow V$ assigns nodes to free variables

“The node x has at least two neighbours”

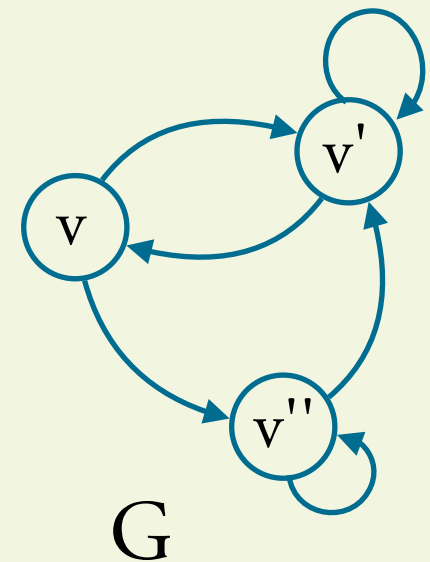
$$\phi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

$$G \models_{\alpha} \phi \quad \text{if} \quad \alpha = \{x \mapsto v\}$$

“Every node has at least two neighbours”

$$\psi = \forall x \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

$$G \models_{\emptyset} \psi$$



Formal Semantics of FO

$G \models_{\alpha} \exists x \phi$ **iff** for **some** $v \in V$ and $\alpha' = \alpha \cup \{x \mapsto v\}$ we have $G \models_{\alpha'} \phi$

$G \models_{\alpha} \forall x \phi$ **iff** for **every** $v \in V$ and $\alpha' = \alpha \cup \{x \mapsto v\}$ we have $G \models_{\alpha'} \phi$

$G \models_{\alpha} \phi \wedge \psi$ **iff** $G \models_{\alpha} \phi$ and $G \models_{\alpha} \psi$

$G \models_{\alpha} \neg \phi$ **iff** it is not true that $G \models_{\alpha} \phi$

$G \models_{\alpha} x=y$ **iff** $\alpha(x)=\alpha(y)$

$G \models_{\alpha} E(x,y)$ **iff** $(\alpha(x),\alpha(y)) \in E$

Formulas as queries

$\phi(x_1, \dots, x_n)$ evaluated on $G=(V,E)$ yields all the bindings that satisfy ϕ :

$$\phi(G) = \{ (\alpha(x_1), \dots, \alpha(x_n)) \mid G \models_{\alpha} \phi, \alpha: \{x_1, \dots, x_n\} \longrightarrow V \}$$

Formulas as queries

$\phi(x_1, \dots, x_n)$ evaluated on $G=(V,E)$ yields all the bindings that satisfy ϕ :

$$\phi(G) = \{ (\alpha(x_1), \dots, \alpha(x_n)) \mid G \models_{\alpha} \phi, \alpha: \{x_1, \dots, x_n\} \longrightarrow V \}$$

“The node x has at least two neighbours”

$$\phi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$



“Return all nodes with
at least two neighbours”

Formulas as queries

$\phi(x_1, \dots, x_n)$ evaluated on $G=(V,E)$ yields all the bindings that satisfy ϕ :

$$\phi(G) = \{ (\alpha(x_1), \dots, \alpha(x_n)) \mid G \models_{\alpha} \phi, \alpha: \{x_1, \dots, x_n\} \longrightarrow V \}$$

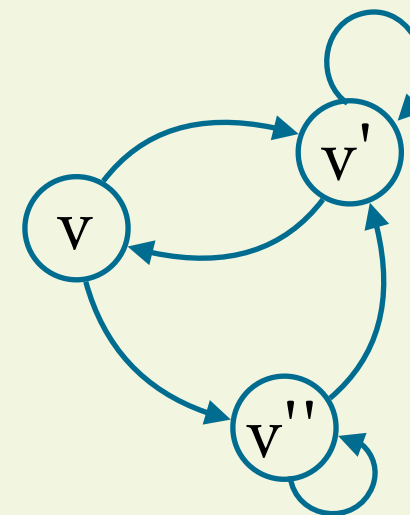
“The node x has at least two neighbours”

$$\phi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

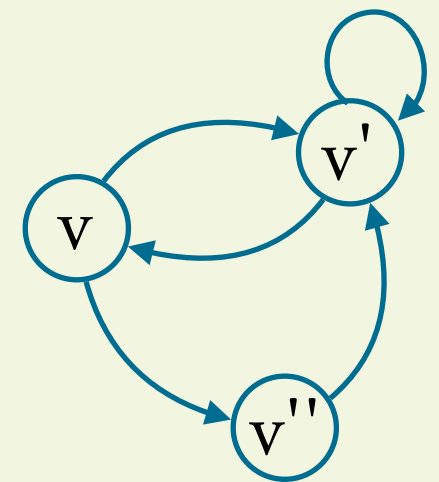
↷ “Return all nodes with at least two neighbours”

$$\phi(G) = \{v, v', v''\}$$

$$\phi(G') = \{v, v'\}$$



G



G'

Formulas as queries

$\phi(x_1, \dots, x_n)$ evaluated on $G=(V,E)$ yields all the bindings that satisfy ϕ :

$$\phi(G) = \{ (\alpha(x_1), \dots, \alpha(x_n)) \mid G \models_{\alpha} \phi, \alpha: \{x_1, \dots, x_n\} \longrightarrow V \}$$

“The node x has at least two neighbours”

$$\phi(x) = \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

↷ “Return all nodes with at least two neighbours”

$$\phi(G) = \{v, v', v''\}$$

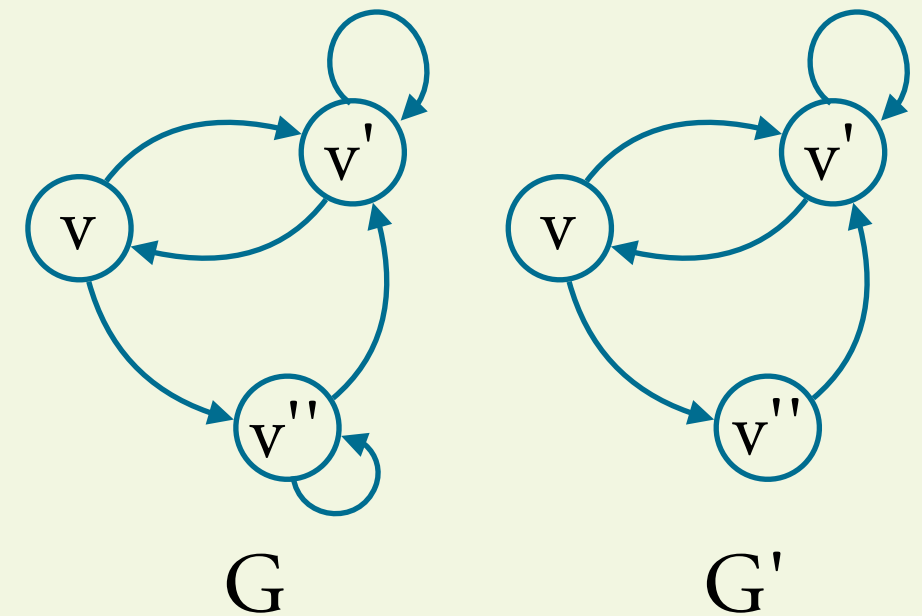
$$\phi(G') = \{v, v'\}$$

“Every node has two neighbours”

$$\psi = \forall x \exists y \exists z (\neg(y=z) \wedge E(x,y) \wedge E(x,z))$$

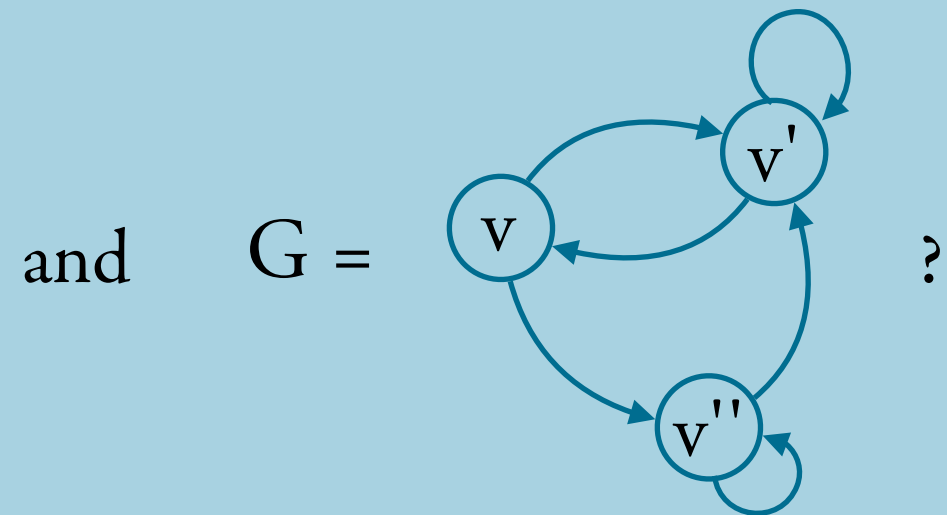
$$\psi(G) = \{()\} \rightsquigarrow \text{set with one element: the 0-tuple}$$

$$\psi(G') = \{\} \rightsquigarrow \text{empty set}$$



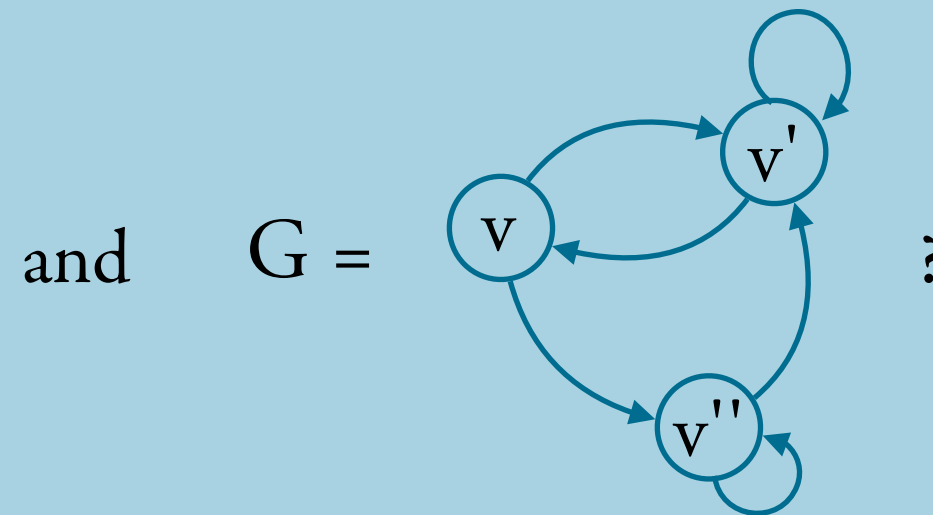
Question: Which bindings α verify $G \models_{\alpha} \phi$ for

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



Question: Which bindings α verify $G \models_{\alpha} \phi$ for

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



Answer: • $\alpha = \{ x \mapsto v, y \mapsto v' \},$

• $\alpha = \{ x \mapsto v, y \mapsto v \},$

• $\alpha = \{ x \mapsto v', y \mapsto v' \},$

• ... and all the rest

$$\phi(G) = \{v, v', v''\} \times \{v, v', v''\}$$

Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations

Queries = Formulas

Rows = Tuples

Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations

Queries = Formulas

Rows = Tuples

Particular to databases:

- Use of constants
- No functions
- **Finite structure**
- Quantification over
active domain

Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations

Queries = Formulas

Rows = Tuples

Particular to databases:

- Use of constants
- No functions
- **Finite structure**
- Quantification over active domain

\models_{finite} is different from \models



Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations

Queries = Formulas

Rows = Tuples

Particular to databases:

- Use of constants
- No functions
- **Finite structure**
- Quantification over active domain

\models_{finite} is different from \models

There are formulas ϕ that are satisfiable only on infinite structures.

Like which?

Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations

Queries = Formulas

Rows = Tuples

Particular to databases:

- Use of constants
- No functions
- **Finite structure**
- Quantification over active domain

\models_{finite} is different from \models

There are formulas ϕ that are satisfiable only on infinite structures.

Like which?

$\phi = \text{“} R(x,y) \text{ is an infinite linear order”}$

Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations

Queries = Formulas

Rows = Tuples

Particular to databases:

- Use of constants
- No functions
- **Finite structure**
- Quantification over active domain

\models_{finite} is different from \models

There are formulas ϕ that are satisfiable only on infinite structures.

Like which?

$\phi = \text{“} R(x,y) \text{ is an infinite linear order”}$

Finite model theory

Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations
Rows = Tuples
Queries = Formulas

[E.F. Codd 1972]

Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations
Rows = Tuples
Queries = Formulas

RA =* FO
How = *What*

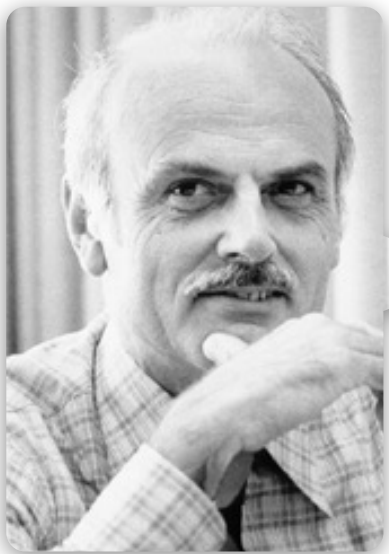
[E.F. Codd 1972]

Formulas as queries

FO can serve as a **declarative** query language on relational databases :
we express the properties of the answer

Tables = Relations
Rows = Tuples
Queries = Formulas

RA =^{*} FO
How = *What*



RA and FO logic have **roughly**^{*} the same expressive power!

^{*}FO without functions, with equality, on finite domains, ...

Formulas as queries

$$\text{RA} \subseteq \text{FO}$$

- $R_1 \times R_2 \quad \rightsquigarrow \quad R_1(x_1, \dots, x_n) \wedge R_2(x_{n+1}, \dots, x_m)$
- $R_1 \cup R_2 \quad \rightsquigarrow \quad R_1(x_1, \dots, x_n) \vee R_2(x_1, \dots, x_n)$
- $\sigma_{\{i_1=j_1, \dots, i_n=j_n\}}(R) \quad \rightsquigarrow \quad R(x_1, \dots, x_m) \wedge (x_{i_1}=x_{j_1}) \wedge \dots \wedge (x_{i_n}=x_{j_n})$
- $\pi_{\{i_1, \dots, i_n\}}(R) \quad \rightsquigarrow \quad \exists(\{x_1, \dots, x_m\} \setminus \{x_{i_1}, \dots, x_{i_n}\}). R(x_1, \dots, x_m)$
- $R_1 \setminus R_2 \quad \rightsquigarrow \quad R_1(x_1, \dots, x_n) \wedge \neg R_2(x_1, \dots, x_n)$
- ...

Formulas as queries

$\text{FO} \subseteq \text{RA}$ does not hold in general!

Formulas as queries

$\text{FO} \subseteq \text{RA}$ does not hold in general!

“the complement of R” $\notin \text{RA}$
 $\in \text{FO} : \neg R(x)$

Formulas as queries

FO $\not\subseteq$ RA

“the complement of R” \notin RA
 \in FO : $\neg R(x)$

Formulas as queries

FO $\not\subseteq$ RA

“the complement of R” \notin RA
 \in FO : $\neg R(x)$

\rightsquigarrow We restrict variables to range over **active domain**

Formulas as queries

FO $\not\subseteq$ RA

“the complement of R” \notin RA
 \in FO : $\neg R(x)$

\leadsto We restrict variables to range over active domain $\dots \blacktriangleright$ elements in the relations

FO_{act}

=

FO restricted
to active domain

Formulas as queries

FO $\not\subseteq$ RA

“the complement of R” \notin RA
 \in FO : $\neg R(x)$

\rightsquigarrow We restrict variables to range over active domain $\dots \blacktriangleright$ elements in the relations

FO^{act}

=

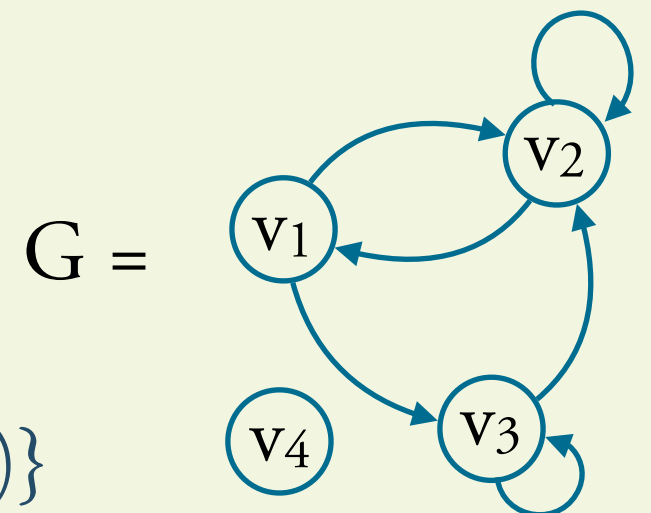
FO restricted
to active domain

$$\phi_1(x) = \forall y E(y, x)$$

$$\phi_1(G) = \{v_2\}$$

$$\phi_2(x, y) = \neg E(x, y)$$

$$\phi_2(G) = \{(v_1, v_1), (v_3, v_1), (v_2, v_3)\}$$



First-order logic **restricted to active domain**

Formal Semantics of **FO_{act}**

$G \models_{\alpha} \exists x \phi$ iff for **some** $v \in \mathbf{ACT}(G)$ and $\alpha' = \alpha \cup \{x \mapsto v\}$ we have $G \models_{\alpha'} \phi$

$G \models_{\alpha} \forall x \phi$ iff for **every** $v \in \mathbf{ACT}(G)$ and $\alpha' = \alpha \cup \{x \mapsto v\}$ we have $G \models_{\alpha'} \phi$

$G \models_{\alpha} \phi \wedge \psi$ iff $G \models_{\alpha} \phi$ and $G \models_{\alpha} \psi$

$G \models_{\alpha} \neg \phi$ iff it is not true that $G \models_{\alpha} \phi$

$G \models_{\alpha} x=y$ iff $\alpha(x)=\alpha(y)$

$G \models_{\alpha} E(x,y)$ iff $(\alpha(x), \alpha(y)) \in E$

$$\mathbf{ACT}(G) = \{v \mid \text{for some } v': (v, v') \in E \text{ or } (v', v) \in E\}$$

First-order logic restricted to active domain

$$\text{FO}^{\text{act}} \subseteq \text{RA}$$

First-order logic restricted to active domain

$$\mathbf{FO}^{\text{act}} \subseteq \mathbf{RA}$$

- Assume:
1. ϕ has variables x_1, \dots, x_n ,
 2. ϕ in normal form: $(\exists^* (\neg \exists)^*)^* +$ quantifier-free $\psi(x_1, \dots, x_n)$

$$\exists x_1 \exists x_2 \neg \exists x_3 \exists x_4 . (E(x_1, x_3) \wedge \neg E(x_4, x_2)) \vee (x_1 = x_3)$$

First-order logic **restricted to active domain**

$$\mathbf{FO}^{\text{act}} \subseteq \mathbf{RA}$$

- Assume:
1. ϕ has variables x_1, \dots, x_n ,
 2. ϕ in normal form: $(\exists^* (\neg \exists)^*)^* +$ quantifier-free $\psi(x_1, \dots, x_n)$

$$\exists x_1 \exists x_2 \neg \exists x_3 \exists x_4 . (E(x_1, x_3) \wedge \neg E(x_4, x_2)) \vee (x_1 = x_3)$$

Adom = RA expression for active domain = “ $\pi_1(E) \cup \pi_2(E)$ ”

- $(R(x_{i_1}, \dots, x_{i_n}))^+ \rightsquigarrow R$
- $(\exists x_i \phi(x_{i_1}, \dots, x_{i_n}))^+ \rightsquigarrow \pi_{\{i_1, \dots, i_n\} \setminus \{i\}}(\phi^+)$
- $(x_i = x_j)^+ \rightsquigarrow \sigma_{\{i=j\}}(\mathbf{Adom} \times \dots \times \mathbf{Adom})$
- $(\psi_1(x_{i_1}, \dots, x_{i_n}) \wedge \psi_2(x_{i_1}, \dots, x_{i_n}))^+ \rightsquigarrow \psi_1^+ \cap \psi_2^+$
- $(\neg \phi(x_{i_1}, \dots, x_{i_n}))^+ \rightsquigarrow \mathbf{Adom} \times \dots \times \mathbf{Adom} \setminus \phi^+$

Translation

First-order logic **restricted to active domain**

$$\mathbf{FO}^{\text{act}} \subseteq \mathbf{RA}$$

- Assume:
1. ϕ has variables x_1, \dots, x_n ,
 2. ϕ in normal form: $(\exists^* (\neg \exists)^*)^* +$ quantifier-free $\psi(x_1, \dots, x_n)$

$$\exists x_1 \exists x_2 \neg \exists x_3 \exists x_4 . (E(x_1, x_3) \wedge \neg E(x_4, x_2)) \vee (x_1 = x_3)$$

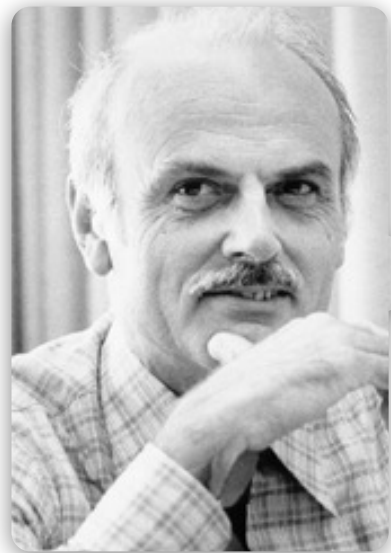
Adom = RA expression for active domain = “ $\pi_1(E) \cup \pi_2(E)$ ”

- $(R(x_{i_1}, \dots, x_{i_n}))^+ \rightsquigarrow R$
- $(\exists x_i \phi(x_{i_1}, \dots, x_{i_n}))^+ \rightsquigarrow \pi_{\{i_1, \dots, i_n\} \setminus \{i\}}(\phi^+)$
- $(x_i = x_j)^+ \rightsquigarrow \sigma_{\{i=j\}}(\mathbf{Adom} \times \dots \times \mathbf{Adom})$
- $(\psi_1(x_{i_1}, \dots, x_{i_n}) \wedge \psi_2(x_{i_1}, \dots, x_{i_n}))^+ \rightsquigarrow \psi_1^+ \cap \psi_2^+$
- $(\neg \phi(x_{i_1}, \dots, x_{i_n}))^+ \rightsquigarrow \mathbf{Adom} \times \dots \times \mathbf{Adom} \setminus \phi^+$

$$A \cap B = (A \cup B) \setminus A \setminus B$$

Translation

Corollary



FO^{act} is equivalent to RA

Question 1: How is $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2))$ expressed in FO?
Remember: R_1, R_2 are binary

Question 2: How is $\exists y, z . (R_1(x, y) \wedge R_1(y, z) \wedge x \neq z)$ expressed in RA?
Remember: The signature is the same as before (R_1, R_2 binary)

- $R_1 \cup R_2$
- $R_1 \times R_2$
- $R_1 \setminus R_2$
- $\sigma_{\{i_1 \neq j_1, \dots, i_n \neq j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$
- $\pi_{\{i_1, \dots, i_n\}}(R) := \{(x_{i_1}, \dots, x_{i_n}) \mid (x_1, \dots, x_m) \in R\}$

Question 1: How is $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2))$ expressed in FO?
Remember: R_1, R_2 are binary

Answer: $\exists x_2 . \left(\exists x_1, x_4 . \left(R_1(x_1, x_2) \wedge R_2(x_1, x_4) \right) \wedge R_2(x_2, x_5) \right)$

Question 2: How is $\exists y, z . (R_1(x, y) \wedge R_1(y, z) \wedge x \neq z)$ expressed in RA?
Remember: The signature is the same as before (R_1, R_2 binary)

- $R_1 \cup R_2$
- $R_1 \times R_2$
- $R_1 \setminus R_2$
- $\sigma_{\{i_1 \neq j_1, \dots, i_n \neq j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$
- $\pi_{\{i_1, \dots, i_n\}}(R) := \{(x_{i_1}, \dots, x_{i_n}) \mid (x_1, \dots, x_m) \in R\}$

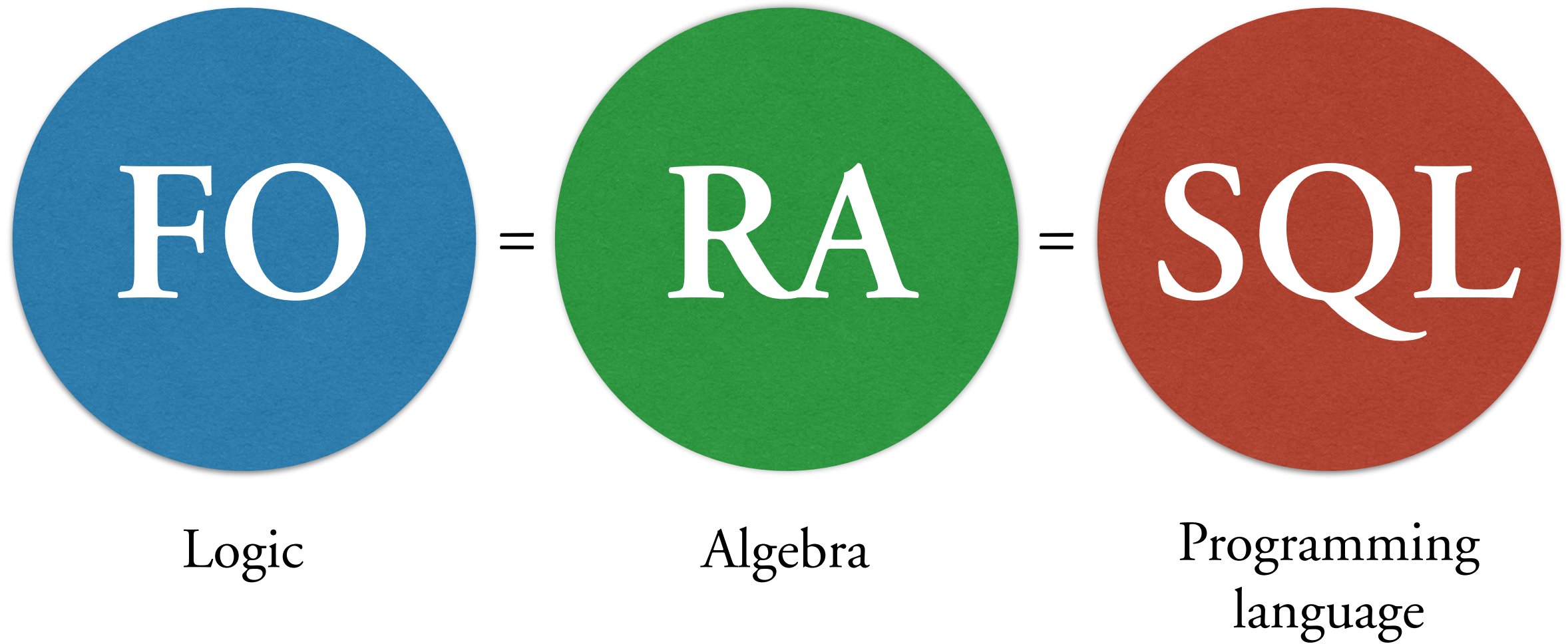
Question 1: How is $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2))$ expressed in FO?
Remember: R_1, R_2 are binary

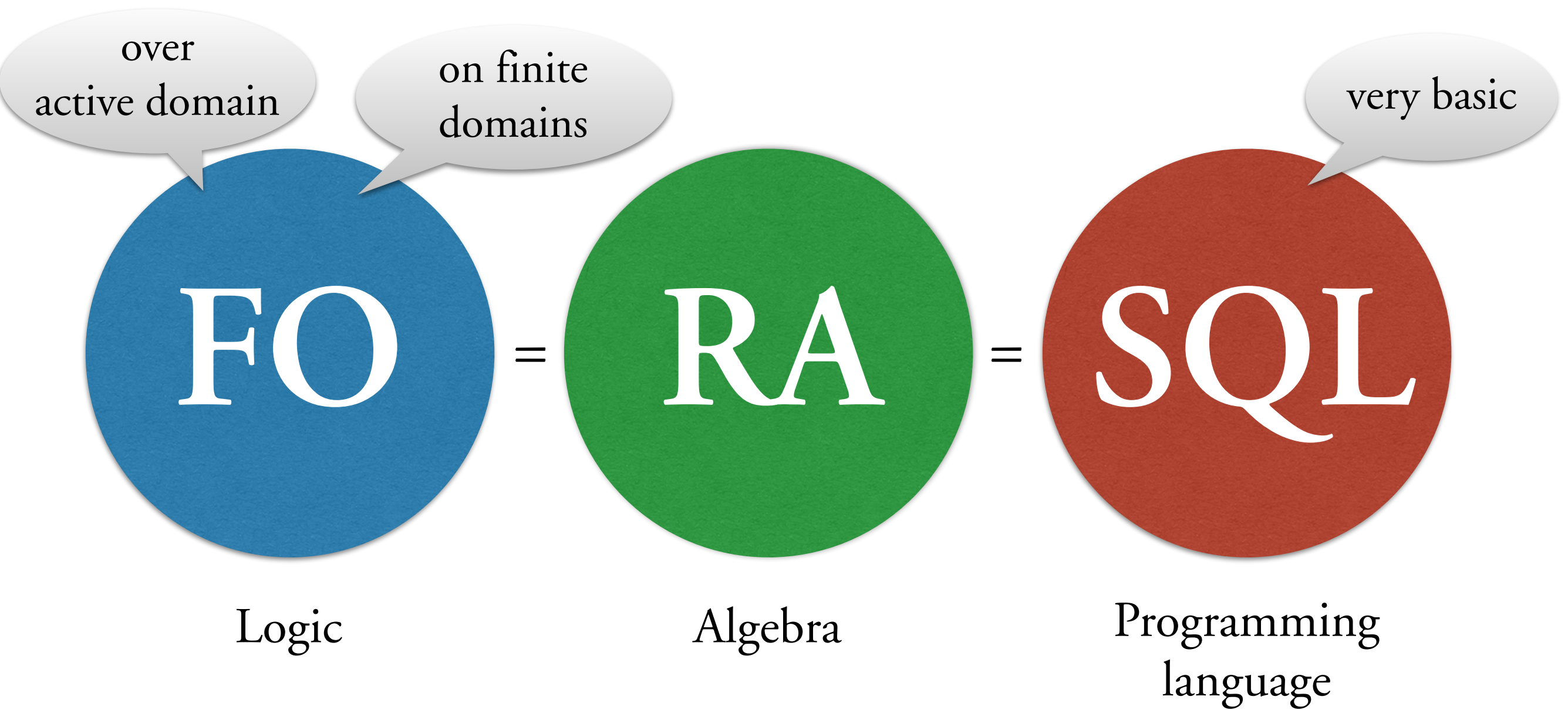
Answer: $\exists x_2 . \left(\exists x_1, x_4 . \left(R_1(x_1, x_2) \wedge R_2(x_1, x_4) \right) \wedge R_2(x_2, x_5) \right)$

Question 2: How is $\exists y, z . (R_1(x, y) \wedge R_1(y, z) \wedge x \neq z)$ expressed in RA?
Remember: The signature is the same as before (R_1, R_2 binary)

- $R_1 \cup R_2$
- $R_1 \times R_2$
- $R_1 \setminus R_2$
- $\sigma_{\{i_1 \neq j_1, \dots, i_n \neq j_n\}}(R) := \{(x_1, \dots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \wedge \dots \wedge (x_{i_n} \neq x_{j_n})\}$
- $\pi_{\{i_1, \dots, i_n\}}(R) := \{(x_{i_1}, \dots, x_{i_n}) \mid (x_1, \dots, x_m) \in R\}$

Answer: $\pi_1(\sigma_{\{2=3, 1 \neq 4\}}(R_1 \times R_1))$





Algorithmic problems for query languages

Evaluation problem: Given a query Q , a database instance db , and a tuple t , is $t \in Q(db)$?

→ How hard is it to retrieve data?

Algorithmic problems for query languages

Evaluation problem: Given a query Q , a database instance db , and a tuple t , is $t \in Q(db)$?

→ How hard is it to retrieve data?

Emptiness problem: Given a query Q , is there a database instance db so that $Q(db) \neq \emptyset$?

→ Does Q make sense? Is it a contradiction? (Query optimization)

Algorithmic problems for query languages

Evaluation problem: Given a query Q , a database instance db , and a tuple t , is $t \in Q(db)$?

↪ How hard is it to retrieve data?

Emptiness problem: Given a query Q , is there a database instance db so that $Q(db) \neq \emptyset$?

↪ Does Q make sense? Is it a contradiction? (Query optimization)

Equivalence problem: Given queries Q_1, Q_2 , is
$$Q_1(db) = Q_2(db)$$

for all database instances db ?

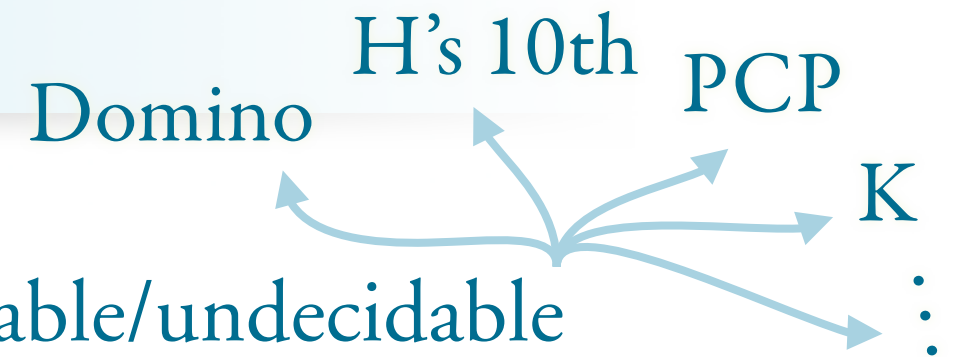
↪ Can we safely replace a query with another? (Query optimization)

Complexity theory

What can be **mechanized**? \leadsto decidable/undecidable

How **hard** is it to mechanise? \leadsto complexity classes

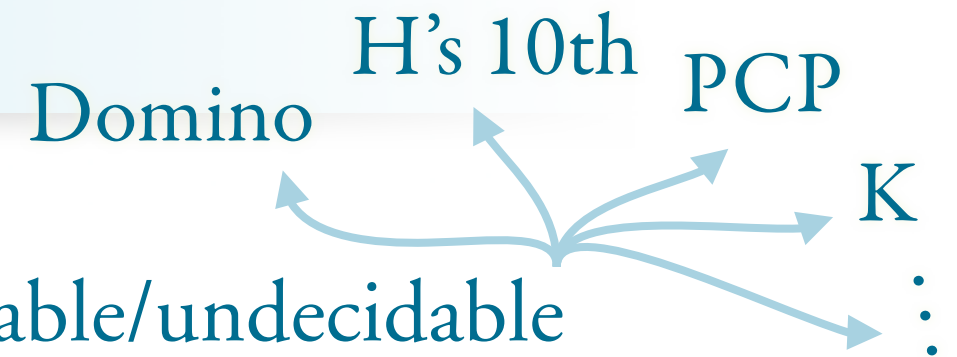
Complexity theory



What can be **mechanized**? \leadsto decidable/undecidable

How **hard** is it to mechanise? \leadsto complexity classes

Complexity theory



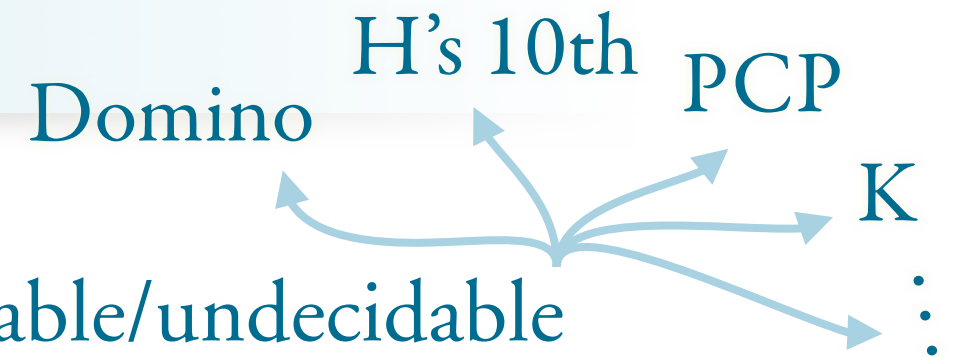
What can be **mechanized**? \leadsto decidable/undecidable

How **hard** is it to mechanise? \leadsto complexity classes

.....► usage of resources:

- time
- memory

Complexity theory



What can be **mechanized**? \leadsto decidable/undecidable

How **hard** is it to mechanise? \leadsto complexity classes

..... \blacktriangleright usage of resources:

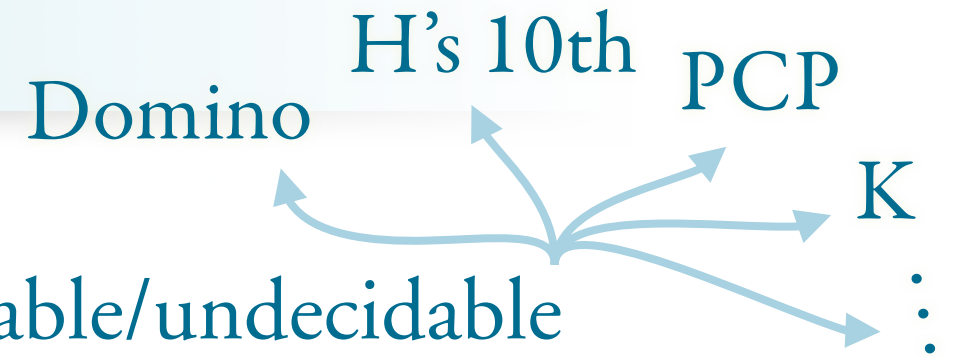
- time
- memory

Algorithm **Alg** is **TIME**-bounded

by a function $f: \mathbb{N} \longrightarrow \mathbb{N}$ if

Alg(*input*) uses less than $f(|input|)$ units of TIME.

Complexity theory



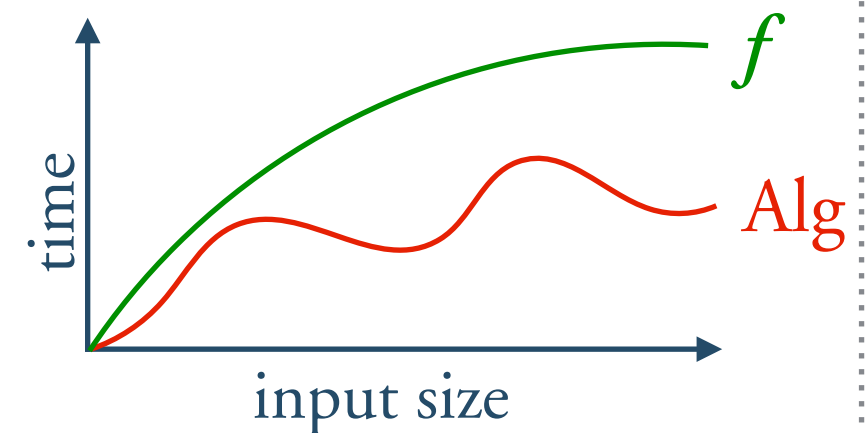
What can be mechanized? \leadsto decidable/undecidable

How **hard** is it to mechanise? \leadsto complexity classes

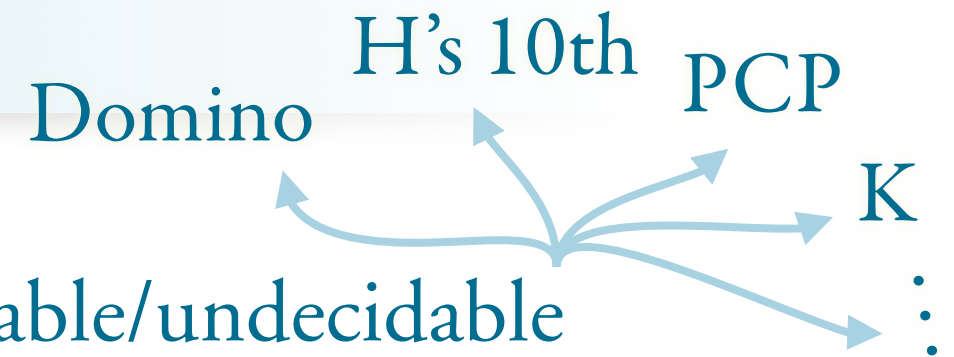
usage of resources:

- time
- memory

Algorithm **Alg** is **TIME**-bounded
by a function $f: \mathbb{N} \longrightarrow \mathbb{N}$ if
Alg(*input*) uses less than $f(|input|)$ units of TIME.



Complexity theory



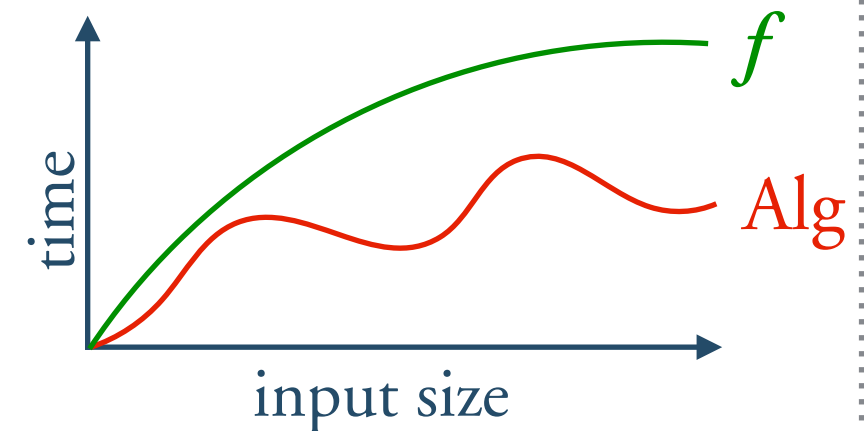
What can be mechanized? \leadsto decidable/undecidable

How hard is it to mechanise? \leadsto complexity classes

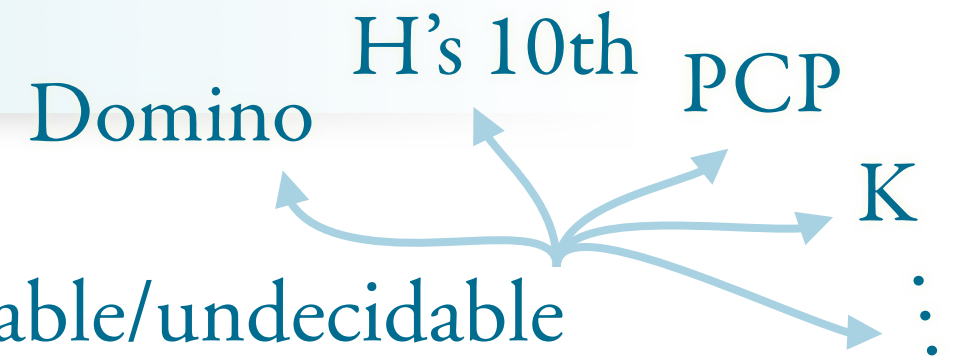
..... \rightarrow usage of resources:

- time
- memory

Algorithm **Alg** is ~~TIME~~^{SPACE}-bounded
by a function $f: \mathbb{N} \rightarrow \mathbb{N}$ if
Alg(*input*) uses less than $f(|input|)$ units of ~~TIME~~^{SPACE}.



Complexity theory



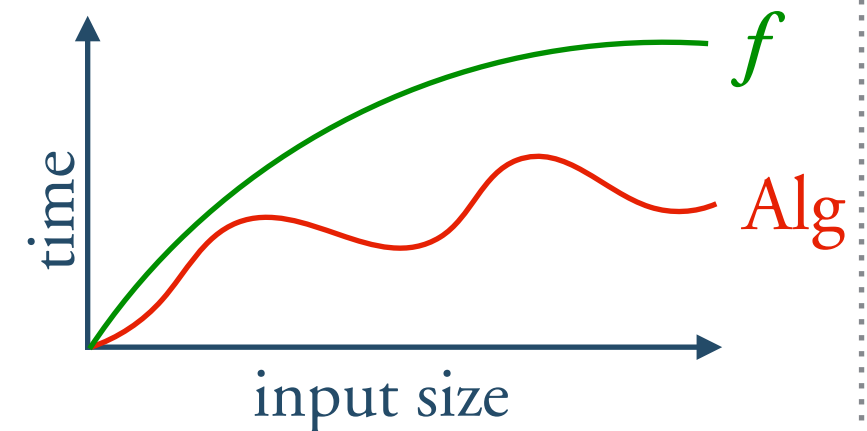
What can be mechanized? \leadsto decidable/undecidable

How hard is it to mechanise? \leadsto complexity classes

..... \rightarrow usage of resources:

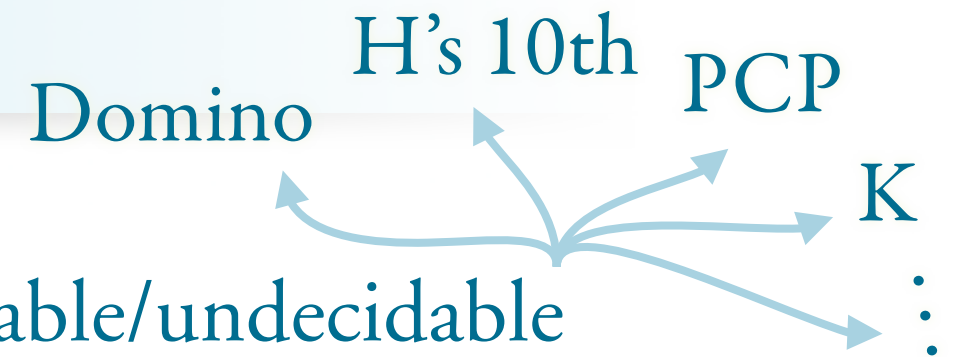
- time
- memory

Algorithm **Alg** is ~~TIME~~^{SPACE}-bounded
by a function $f: \mathbb{N} \rightarrow \mathbb{N}$ if
Alg(*input*) uses less than $f(|input|)$ units of ~~TIME~~^{SPACE}.



$\text{LOGSPACE} \subsetneq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \dots$

Complexity theory



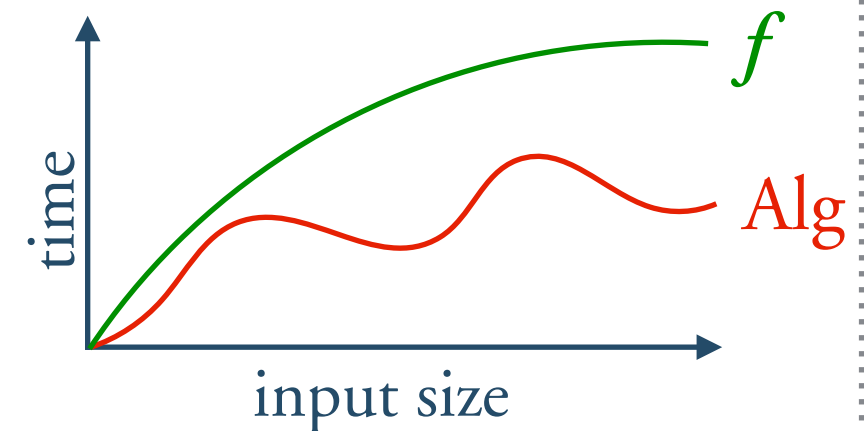
What can be **mechanized**? \leadsto decidable/undecidable

How **hard** is it to mechanise? \leadsto complexity classes

..... \rightarrow usage of resources:

- time
- memory

Algorithm **Alg** is ~~TIME~~^{SPACE}-bounded
by a function $f: \mathbb{N} \rightarrow \mathbb{N}$ if
Alg(*input*) uses less than $f(|input|)$ units of ~~TIME~~^{SPACE}.



\rightarrow TIME-bounded by a polynomial

$\text{LOGSPACE} \subsetneq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \dots$

\rightarrow SPACE-bounded by a polynomial

\rightarrow SPACE-bounded by $\log(n)$

Algorithmic problems for FO

Evaluation problem: Given a FO formula $\phi(x_1, \dots, x_n)$, a graph G , and a binding α , does $G \models_{\alpha} \phi$?

Satisfiability problem: Given a FO formula ϕ , is there a graph G and binding α , such that $G \models_{\alpha} \phi$?

Equivalence problem: Given FO formulae ϕ, ψ , is
$$G \models_{\alpha} \phi \text{ iff } G \models_{\alpha} \psi$$

for all graphs G and bindings α ?

Algorithmic problems for FO

Evaluation problem: Given a FO formula $\phi(x_1, \dots, x_n)$, a graph G , and a binding α , does $G \models_{\alpha} \phi$?

DECIDABLE \rightsquigarrow foundations of the database industry

Satisfiability problem: Given a FO formula ϕ , is there a graph G and binding α , such that $G \models_{\alpha} \phi$?

Equivalence problem: Given FO formulae ϕ, ψ , is
$$G \models_{\alpha} \phi \text{ iff } G \models_{\alpha} \psi$$

for all graphs G and bindings α ?

Algorithmic problems for FO

Evaluation problem: Given a FO formula $\phi(x_1, \dots, x_n)$, a graph G , and a binding α , does $G \models_{\alpha} \phi$?

DECIDABLE \rightsquigarrow foundations of the database industry

Satisfiability problem: Given a FO formula ϕ , is there a graph G and binding α , such that $G \models_{\alpha} \phi$?

 **UNDECIDABLE** \rightsquigarrow both for \models and \models_{finite}

Equivalence problem: Given FO formulae ϕ, ψ , is
$$G \models_{\alpha} \phi \text{ iff } G \models_{\alpha} \psi$$

for all graphs G and bindings α ?

Algorithmic problems for FO

Evaluation problem: Given a FO formula $\phi(x_1, \dots, x_n)$, a graph G , and a binding α , does $G \models_{\alpha} \phi$?

DECIDABLE \rightsquigarrow foundations of the database industry

Satisfiability problem: Given a FO formula ϕ , is there a graph G and binding α , such that $G \models_{\alpha} \phi$?

 **UNDECIDABLE** \rightsquigarrow both for \models and \models_{finite}

Equivalence problem: Given FO formulae ϕ, ψ , is
$$G \models_{\alpha} \phi \text{ iff } G \models_{\alpha} \psi$$

for all graphs G and bindings α ?

 **UNDECIDABLE** \rightsquigarrow by reduction to the satisfiability problem

Algorithmic problems for FO

Satisfiability problem: Given a FO formula ϕ , is there a graph G and binding α , such that $G \models_{\alpha} \phi$?

 UNDECIDABLE \rightsquigarrow both for \models and \models_{finite} [Trakhtenbrot '50]

Algorithmic problems for FO

Satisfiability problem: Given a FO formula ϕ , is there a graph G and binding α , such that $G \models_{\alpha} \phi$?

 UNDECIDABLE \rightsquigarrow both for \models and \models_{finite} [Trakhtenbrot '50]

Proof: By reduction from the Domino (aka Tiling) problem.

Algorithmic problems for FO

Satisfiability problem: Given a FO formula ϕ , is there a graph G and binding α , such that $G \models_{\alpha} \phi$?

👁 UNDECIDABLE \rightsquigarrow both for \models and \models_{finite} [Trakhtenbrot '50]

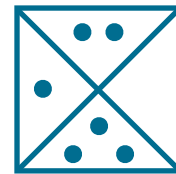
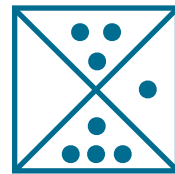
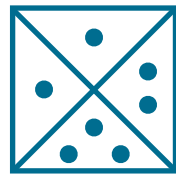
Proof: By reduction from the Domino (aka Tiling) problem.

Reduction from P to P' : Algorithm that solves P using a $O(1)$ procedure
“ $P'(x)$ ”
that returns the truth value of $P'(x)$.

The (undecidable) Domino problem

Domino

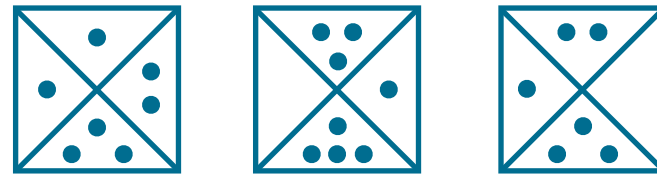
Input: 4-sided dominos:



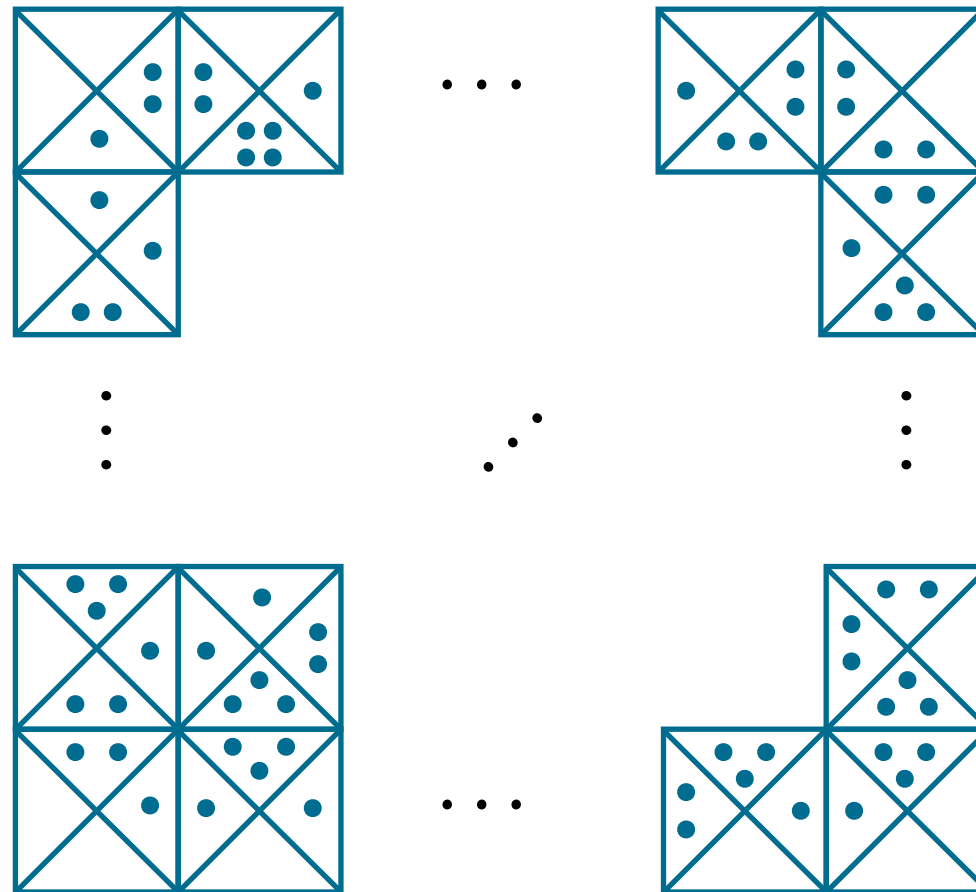
The (undecidable) Domino problem

Domino

Input: 4-sided dominos:



Output: Is it possible to form a white-bordered rectangle? (of any size)

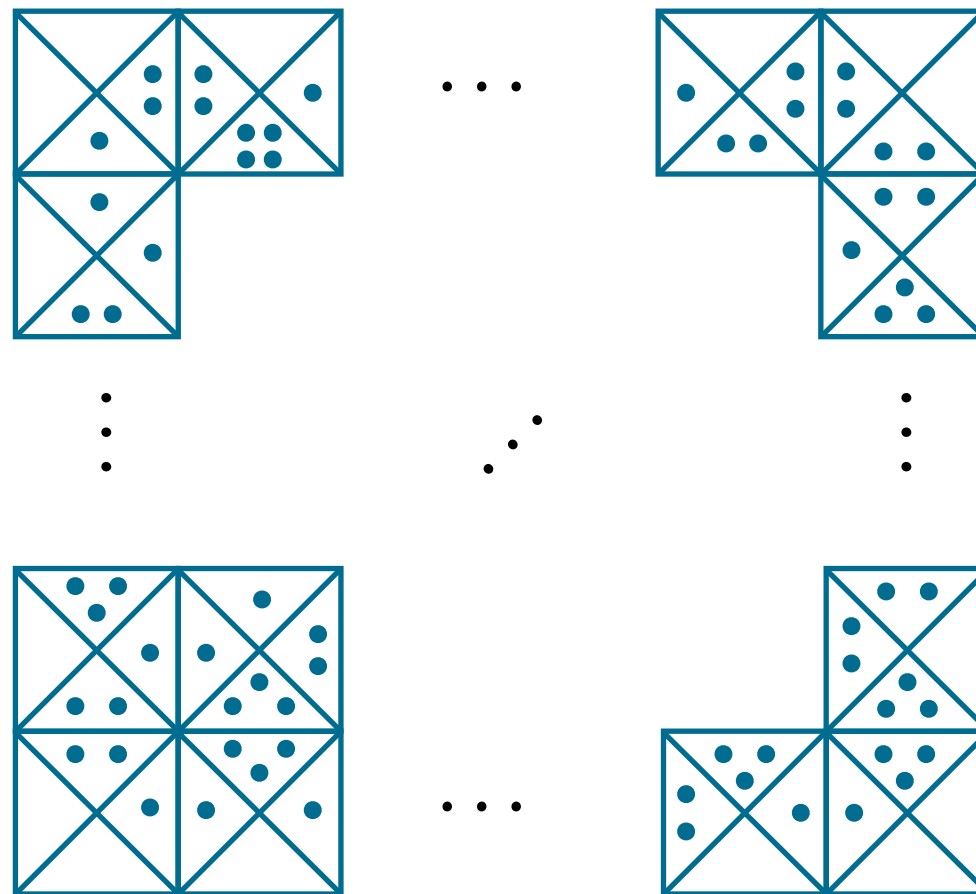


The (undecidable) Domino problem

Domino

Input: 4-sided dominos: 

Output: Is it possible to form a white-bordered rectangle? (of any size)

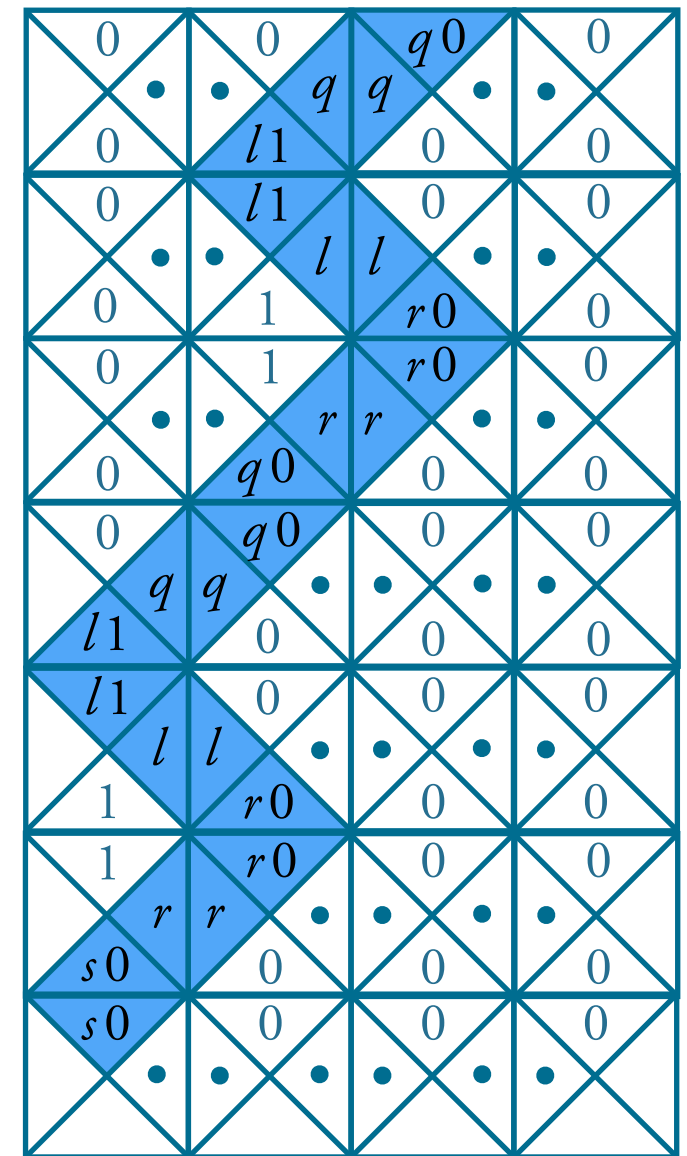


Rules: sides must match,
you can't rotate the dominos, but you can 'clone' them.

The (undecidable) Domino problem

Domino - Why is it undecidable?

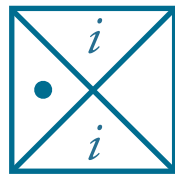
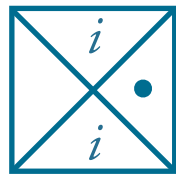
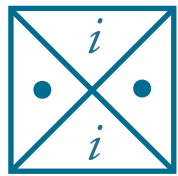
It can easily encode *halting* computations of Turing machines:



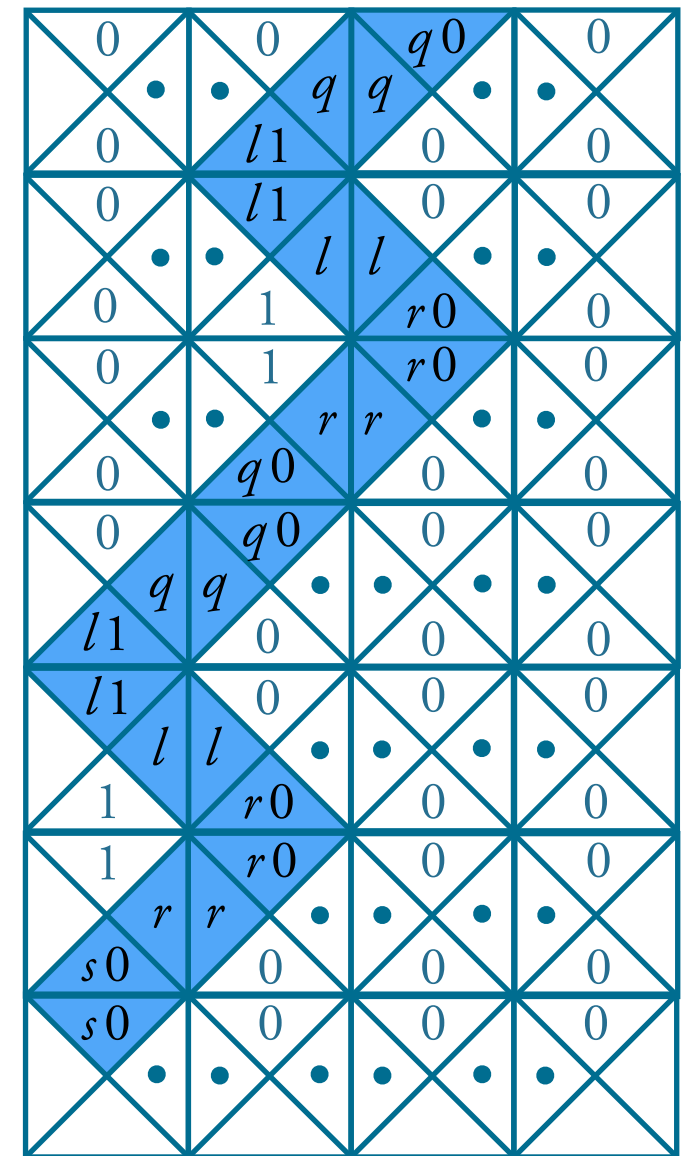
The (undecidable) Domino problem

Domino - Why is it undecidable?

It can easily encode *halting* computations of Turing machines:



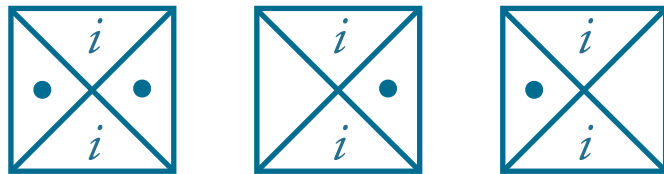
(head is elsewhere,
symbol is not modified)



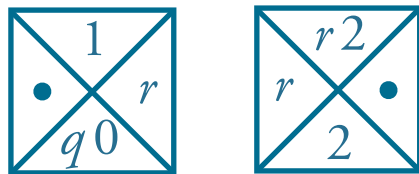
The (undecidable) Domino problem

Domino - Why is it undecidable?

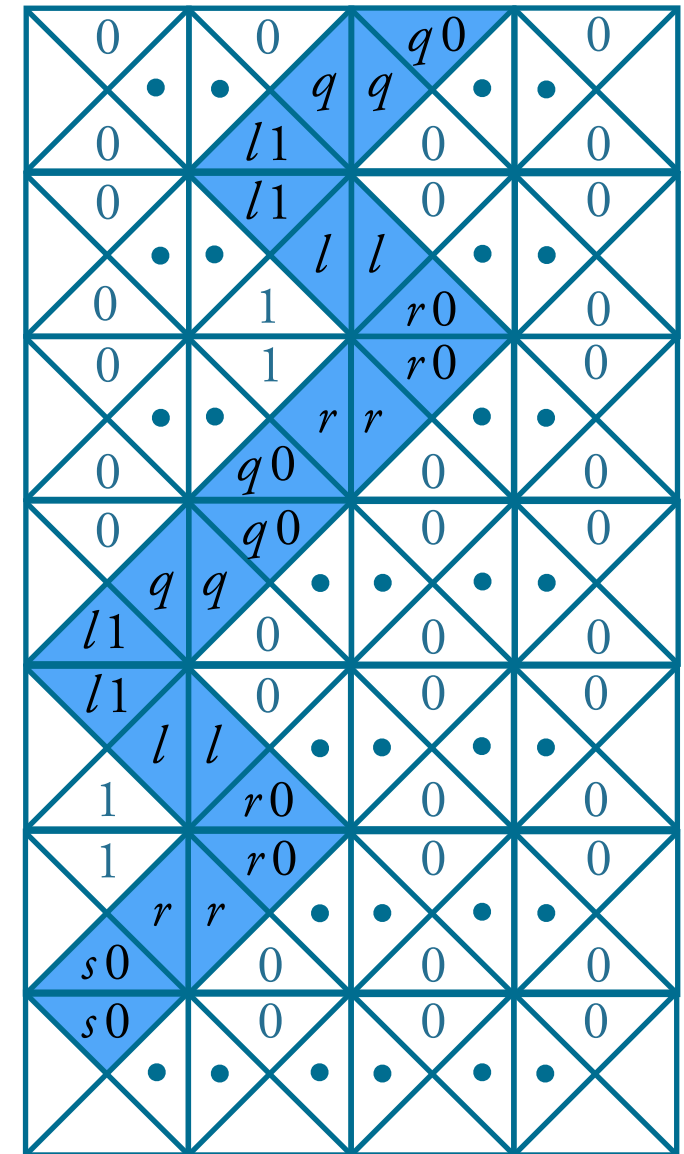
It can easily encode *halting* computations of Turing machines:



(head is elsewhere,
symbol is not modified)



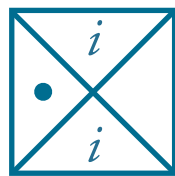
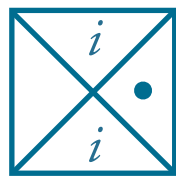
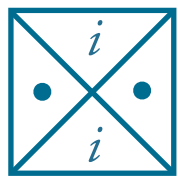
(head is here, symbol is
rewritten, head moves right)



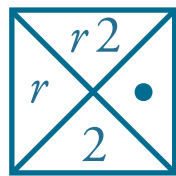
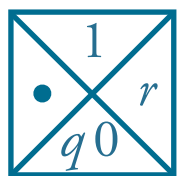
The (undecidable) Domino problem

Domino - Why is it undecidable?

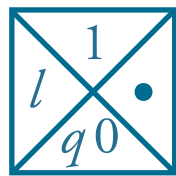
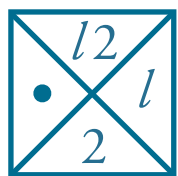
It can easily encode *halting* computations of Turing machines:



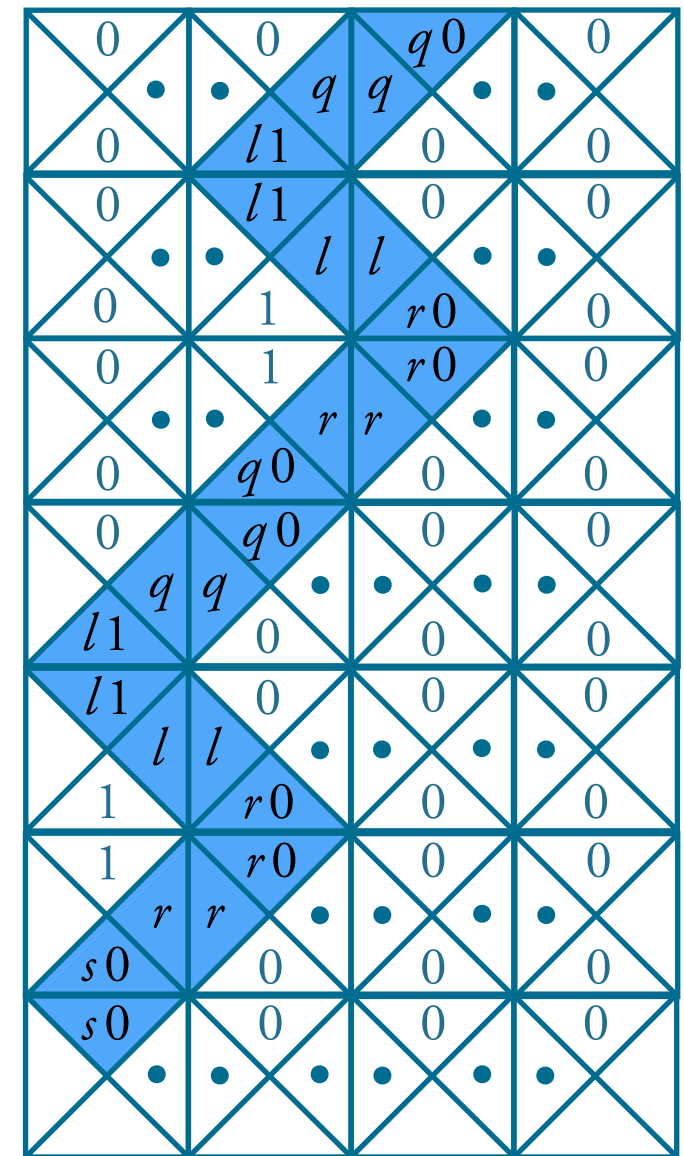
(head is elsewhere,
symbol is not modified)



(head is here, symbol is
rewritten, head moves right)



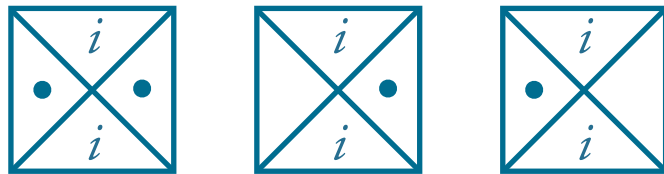
(head is here, symbol is
rewritten, head moves left)



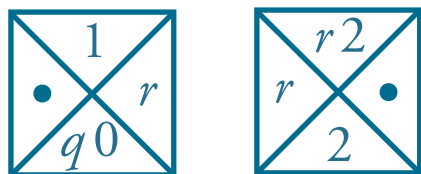
The (undecidable) Domino problem

Domino - Why is it undecidable?

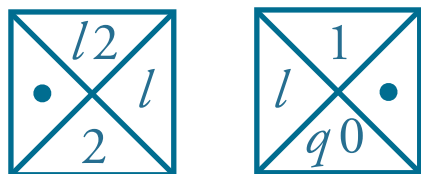
It can easily encode *halting* computations of Turing machines:



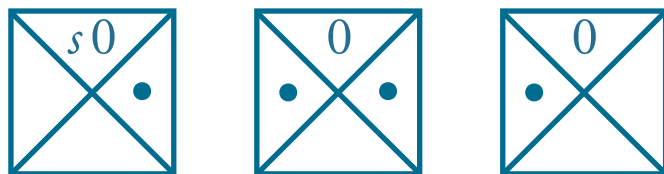
(head is elsewhere,
symbol is not modified)



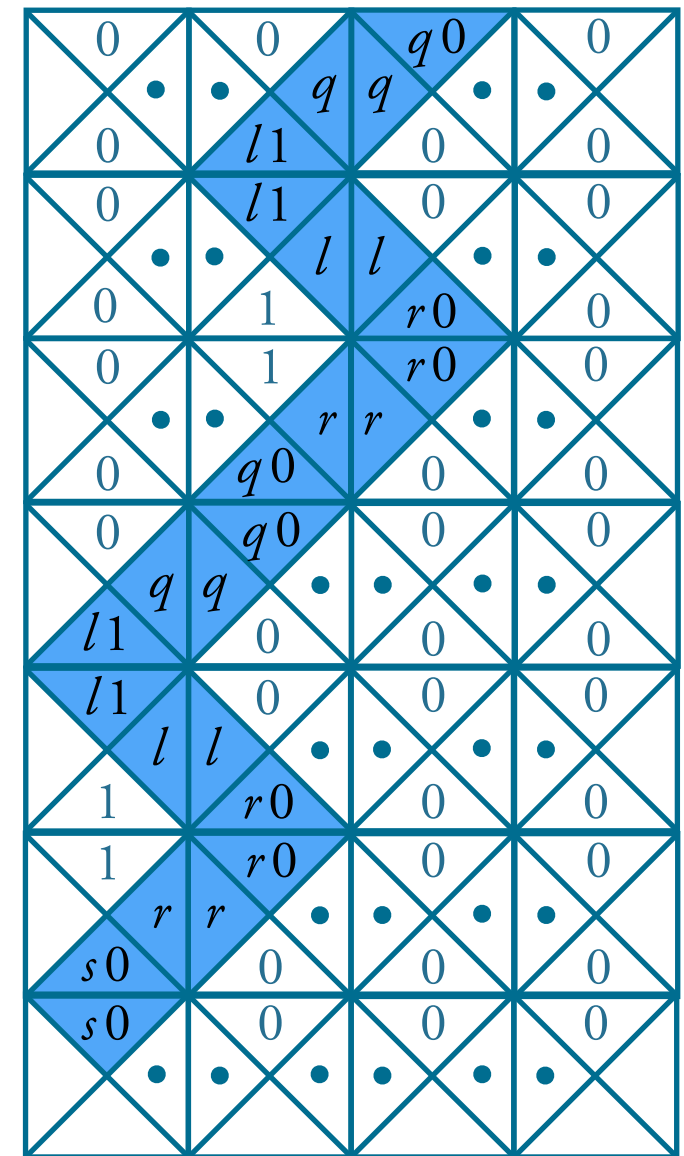
(head is here, symbol is
rewritten, head moves right)



(head is here, symbol is
rewritten, head moves left)



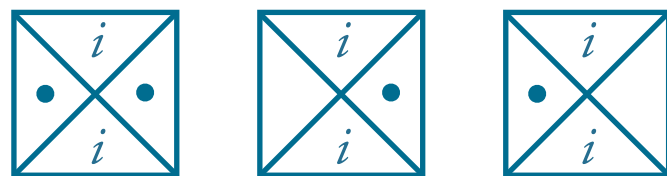
(initial configuration)



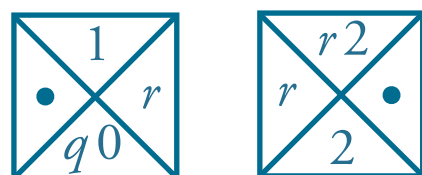
The (undecidable) Domino problem

Domino - Why is it undecidable?

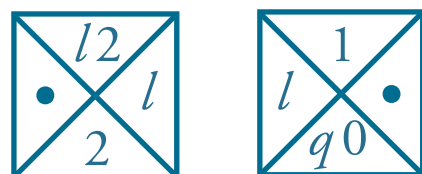
It can easily encode *halting* computations of Turing machines:



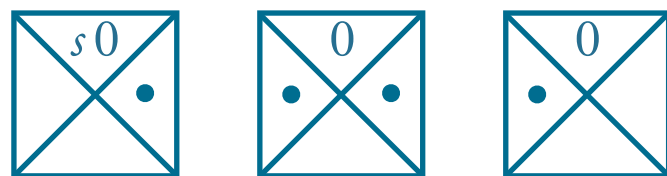
(head is elsewhere,
symbol is not modified)



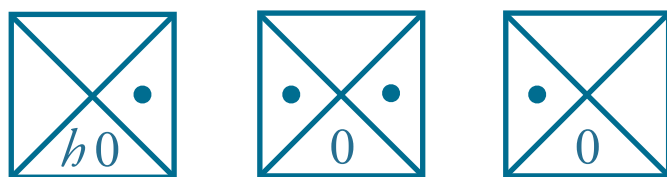
(head is here, symbol is
rewritten, head moves right)



(head is here, symbol is
rewritten, head moves left)

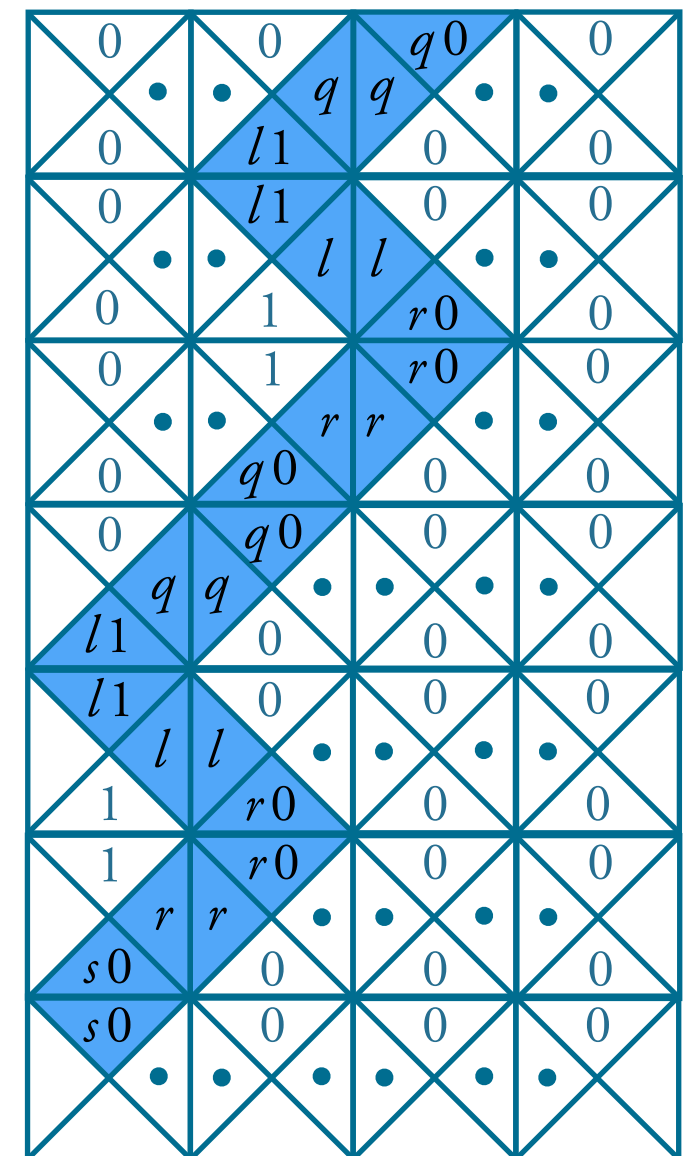


(initial configuration)



(halting configuration)

...

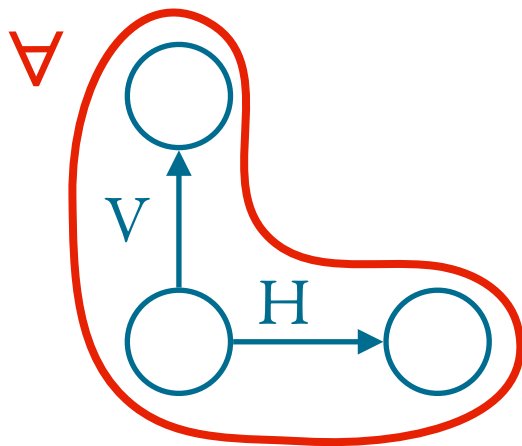


Domino \nleftrightarrow Sat-FO (domino has a solution iff ϕ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...

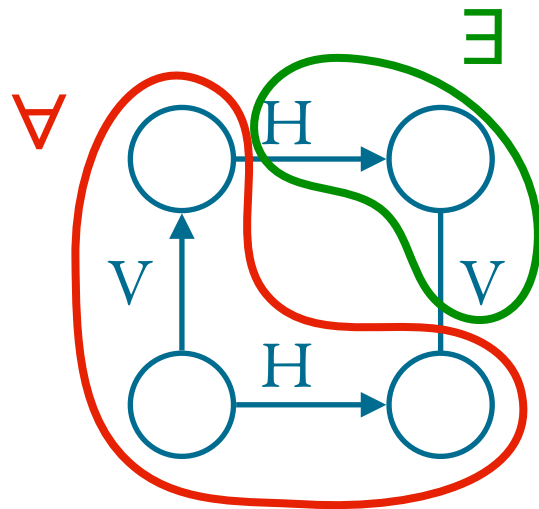
Domino \rightsquigarrow Sat-FO (domino has a solution iff ϕ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...



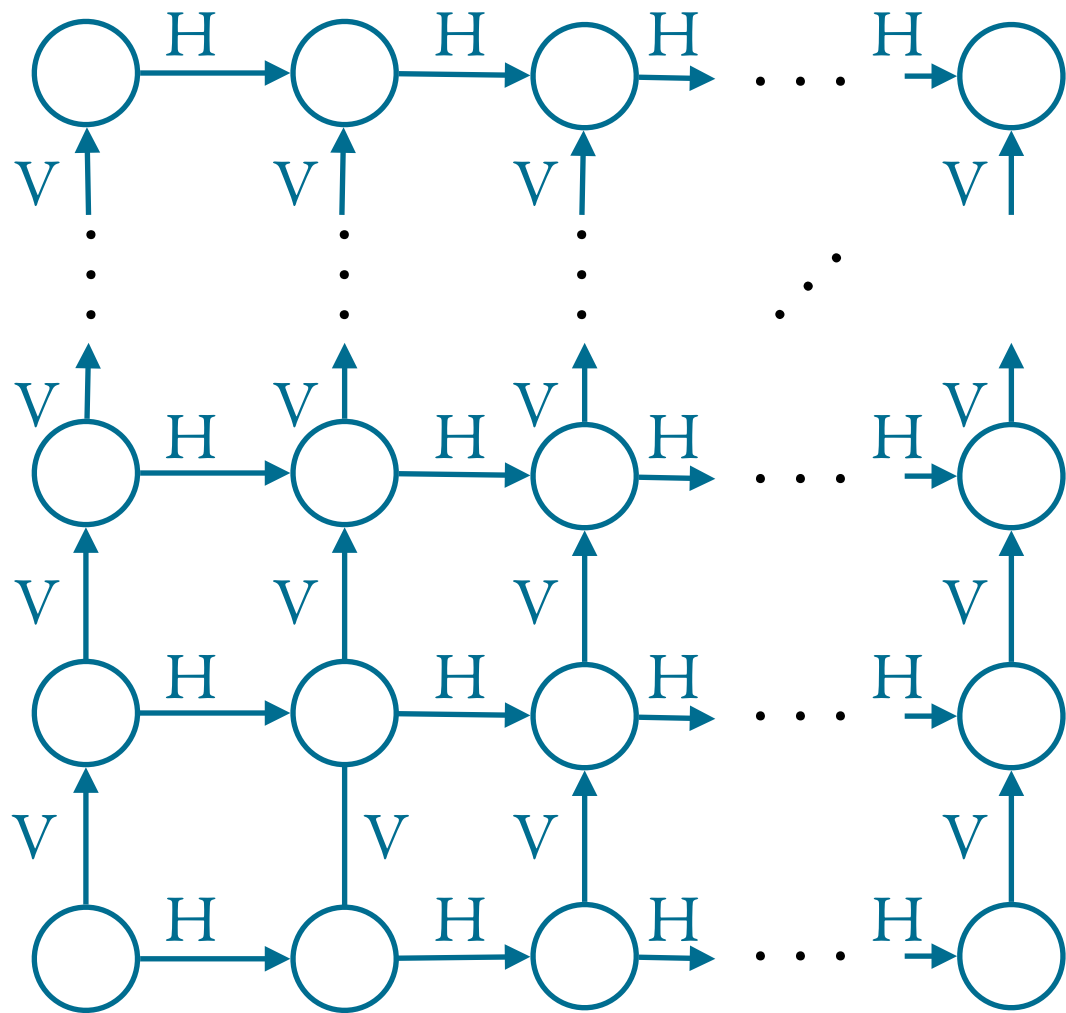
Domino \rightsquigarrow Sat-FO (domino has a solution iff ϕ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...



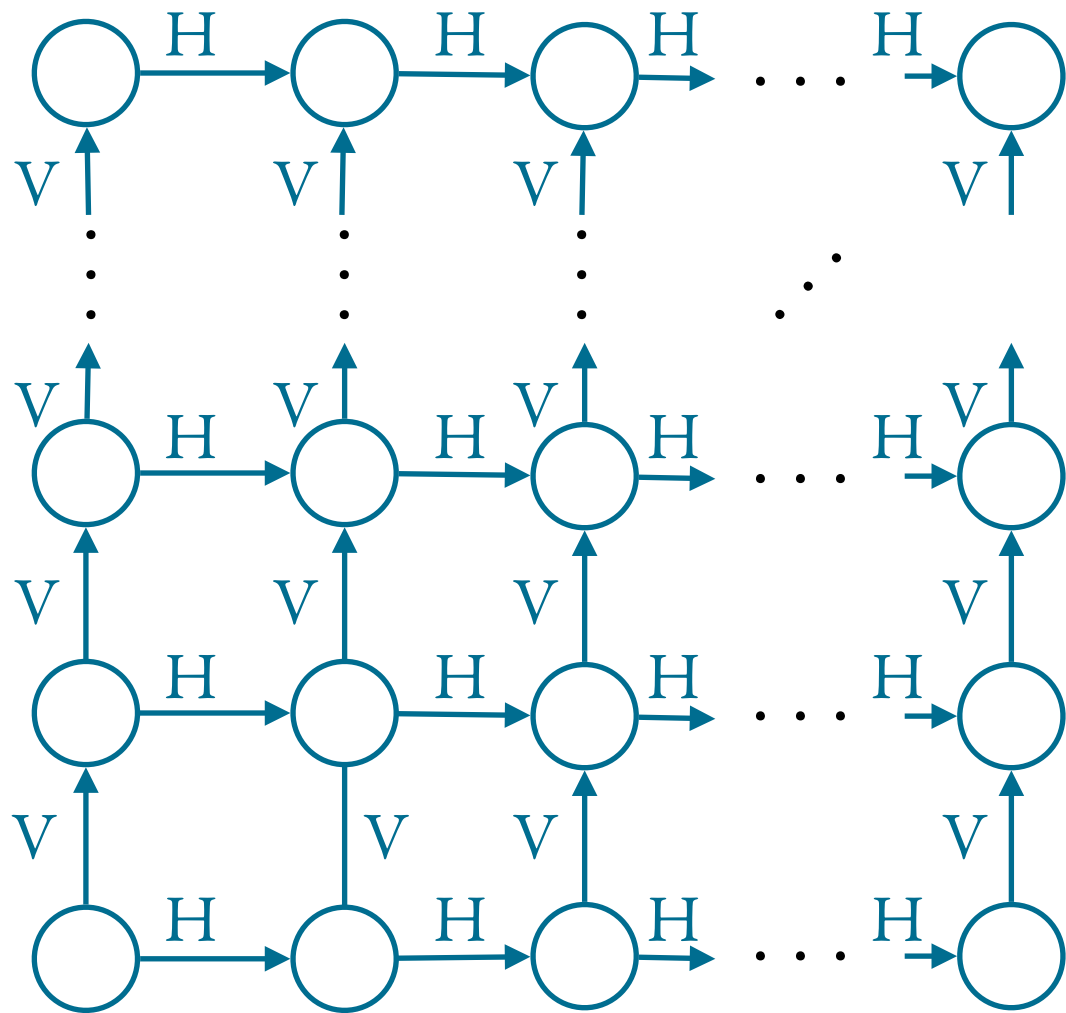
Domino \rightsquigarrow Sat-FO (domino has a solution iff ϕ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...



Domino \rightsquigarrow Sat-FO (domino has a solution iff ϕ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...



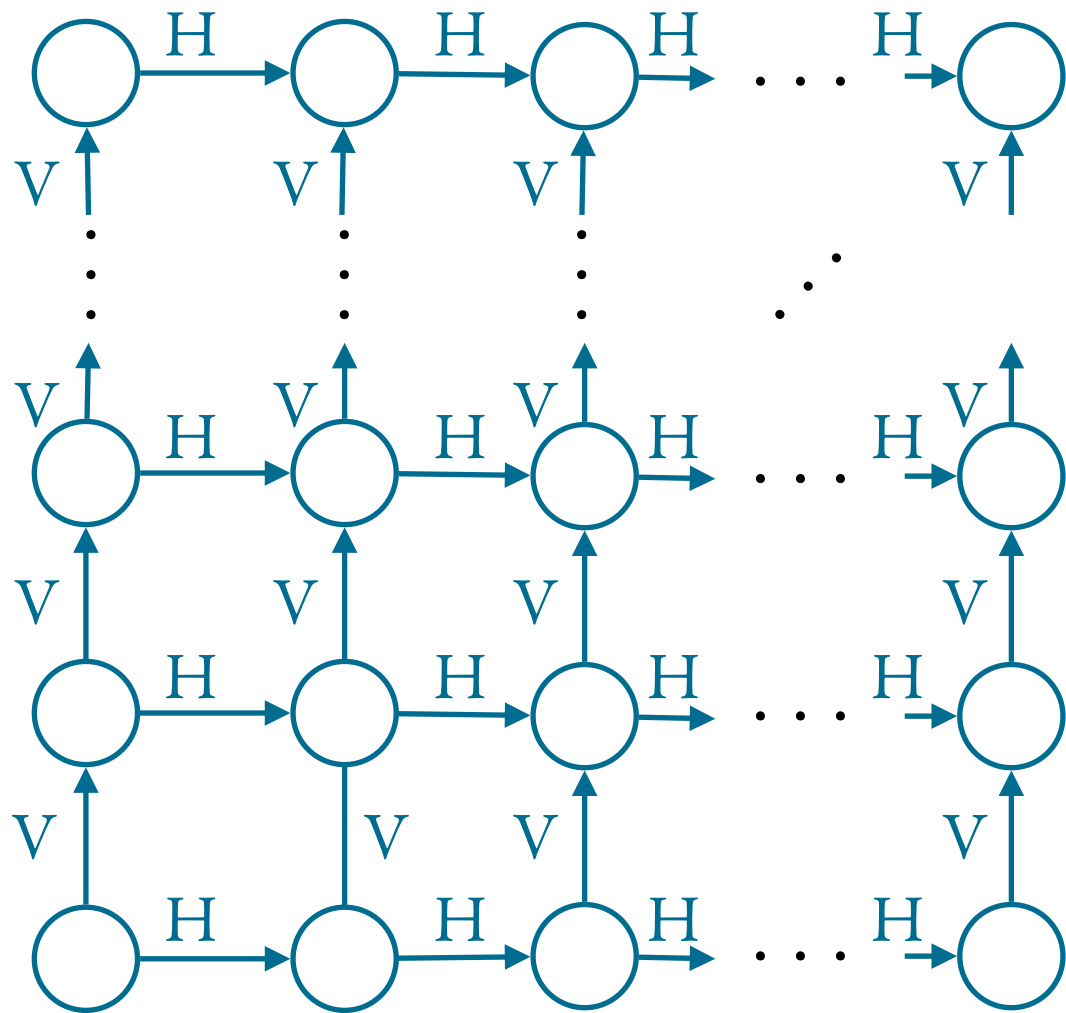
2. Assign one domino to each node:
a unary relation

$$D_{\boxed{\begin{smallmatrix} \cdot & \cdot & \cdot \\ \cdot & \times & \cdot \\ \cdot & \cdot & \cdot \end{smallmatrix}}}(\mathbf{x})$$

for each domino $\boxed{\begin{smallmatrix} \cdot & \cdot & \cdot \\ \cdot & \times & \cdot \\ \cdot & \cdot & \cdot \end{smallmatrix}}$

Domino \rightsquigarrow Sat-FO (domino has a solution iff ϕ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...



2. Assign one domino to each node:
a unary relation

$$D_{\boxed{\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix}}}(\mathbf{x})$$

for each domino $\boxed{\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix}}$

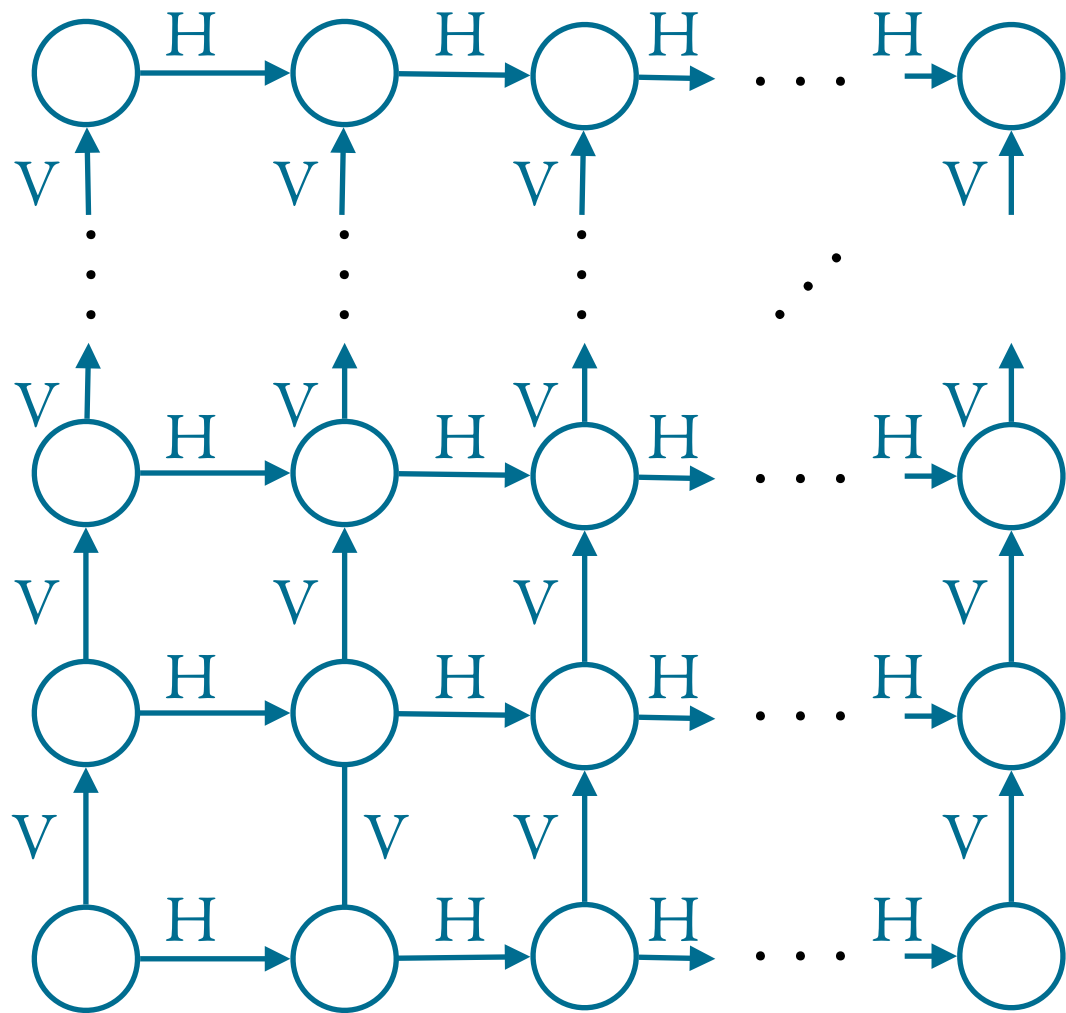
3. Match the sides $\forall \mathbf{x}, \mathbf{y}$

if $H(\mathbf{x}, \mathbf{y})$, then $D_a(\mathbf{x}) \wedge D_b(\mathbf{y})$

for some dominos \mathbf{a}, \mathbf{b} that 'match'
horizontally (Idem vertically)

Domino \rightsquigarrow Sat-FO (domino has a solution iff ϕ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...



2. Assign one domino to each node:
a unary relation

$$D_{\boxed{\begin{smallmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{smallmatrix}}}(\mathbf{x})$$

for each domino $\boxed{\begin{smallmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{smallmatrix}}$

3. Match the sides $\forall x,y$

if $H(x,y)$, then $D_a(x) \wedge D_b(y)$

for some dominos a,b that 'match'
horizontally (Idem vertically)

4. Borders are white.

Algorithmic problems for FO

Evaluation problem: Given a FO formula $\phi(x_1, \dots, x_n)$, a graph G , and a binding α , does $G \models_{\alpha} \phi$?

DECIDABLE \rightsquigarrow foundations of the database industry

Satisfiability problem: Given a FO formula ϕ , is there a graph G and binding α , such that $G \models_{\alpha} \phi$?

 **UNDECIDABLE** \rightsquigarrow both for \models and \models_{finite}

Equivalence problem: Given FO formulae ϕ, ψ , is
$$G \models_{\alpha} \phi \text{ iff } G \models_{\alpha} \psi$$

for all graphs G and bindings α ?

 **UNDECIDABLE** \rightsquigarrow by reduction to the satisfiability problem

Algorithmic problems for FO

Equivalence problem: Given FO formulae ϕ, ψ , is
 $G \models_{\alpha} \phi$ iff $G \models_{\alpha} \psi$
for all graphs G and bindings α ?



UNDECIDABLE \rightsquigarrow by reduction from the satisfiability problem

Algorithmic problems for FO

ϕ is satisfiable iff ϕ is not equivalent to \perp

Satisfiability problem undecidable \rightsquigarrow Equivalence problem undecidable

Equivalence problem: Given FO formulae ϕ, ψ , is
 $G \models_{\alpha} \phi$ iff $G \models_{\alpha} \psi$
for all graphs G and bindings α ?



UNDECIDABLE \rightsquigarrow by reduction from the satisfiability problem

Algorithmic problems for FO

ϕ is **satisfiable** iff ϕ is **not equivalent** to \perp

Satisfiability problem undecidable \rightsquigarrow Equivalence problem undecidable

Actually, there are reductions in both senses:

$\phi(x_1, \dots, x_n)$ and $\psi(y_1, \dots, y_m)$ are **equivalent** iff

- $n=m$
- $(x_1=y_1) \wedge \dots \wedge (x_n=y_n) \wedge \phi(x_1, \dots, x_n) \wedge \neg\psi(y_1, \dots, y_n)$ is unsatisfiable
- $(x_1=y_1) \wedge \dots \wedge (x_n=y_n) \wedge \psi(x_1, \dots, x_n) \wedge \neg\phi(y_1, \dots, y_n)$ is unsatisfiable

Equivalence problem: Given FO formulae ϕ, ψ , is
 $G \models_{\alpha} \phi$ iff $G \models_{\alpha} \psi$
for all graphs G and bindings α ?



UNDECIDABLE \rightsquigarrow by reduction from the satisfiability problem

Algorithmic problems for FO

Evaluation problem: Given a FO formula $\phi(x_1, \dots, x_n)$,
a graph G , and a binding α , does $G \models_{\alpha} \phi$?

DECIDABLE \rightsquigarrow foundations of the database industry

Satisfiability problem: Given a FO formula ϕ , is there a graph G
and binding α , such that $G \models_{\alpha} \phi$?

 **UNDECIDABLE** \rightsquigarrow both for \models and \models_{finite}

Equivalence problem: Given FO formulae ϕ, ψ , is
 $G \models_{\alpha} \phi$ iff $G \models_{\alpha} \psi$
for all graphs G and bindings α ?

 **UNDECIDABLE** \rightsquigarrow by reduction to the satisfiability problem

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$ **Output:** $G \models_{\alpha} \phi ?$

Encoding of $G = (V, E)$

- each node is coded with a bit string of size $\log(|V|)$,
- edge set is encoded by its tuples, e.g. $(100, 101), (010, 010), \dots$

Cost of coding: $\|G\| = |E| \cdot 2 \cdot \log(|V|) \approx |V| \text{ (mod a polynomial)}$

Evaluation problem for FO

$$\text{Input: } \begin{pmatrix} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{pmatrix} \qquad \text{Output: } G \models_{\alpha} \phi ?$$

Encoding of $G = (V, E)$

- each node is coded with a bit string of size $\log(|V|)$,
- edge set is encoded by its tuples, e.g. $(100, 101), (010, 010), \dots$

Cost of coding: $\|G\| = |E| \cdot 2 \cdot \log(|V|) \approx |V|$ (mod a polynomial)

Encoding of $\alpha = \{x_1, \dots, x_n\} \longrightarrow V$

- each node is coded with a bit string of size $\log(|V|)$,

Cost of coding: $\|\alpha\| = n \cdot \log(|V|)$

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

- If $\phi(x_1, \dots, x_n) = E(x_i, x_j)$:
answer YES iff $(\alpha(x_i), \alpha(x_j)) \in E$
- If $\phi(x_1, \dots, x_n) = \psi(x_1, \dots, x_n) \wedge \psi'(x_1, \dots, x_n)$:
answer YES iff $G \models_{\alpha} \psi$ and $G \models_{\alpha} \psi'$
- If $\phi(x_1, \dots, x_n) = \neg \psi(x_1, \dots, x_n)$:
answer NO iff $G \models_{\alpha} \psi$
- If $\phi(x_1, \dots, x_n) = \exists y . \psi(x_1, \dots, x_n, y)$:
answer YES iff for some $v \in V$ and $\alpha' = \alpha \cup \{y \mapsto v\}$
we have $G \models_{\alpha'} \psi$.

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

- If $\phi(x_1, \dots, x_n) = E(x_i, x_j)$:
answer YES iff $(\alpha(x_i), \alpha(x_j)) \in E$
- If $\phi(x_1, \dots, x_n) = \psi(x_1, \dots, x_n) \wedge \psi'(x_1, \dots, x_n)$:
answer YES iff $G \models_{\alpha} \psi$ and $G \models_{\alpha} \psi'$
- If $\phi(x_1, \dots, x_n) = \neg \psi(x_1, \dots, x_n)$:
answer NO iff $G \models_{\alpha} \psi$
- If $\phi(x_1, \dots, x_n) = \exists y . \psi(x_1, \dots, x_n, y)$:
answer YES iff for some $v \in V$ and $\alpha' = \alpha \cup \{y \mapsto v\}$
we have $G \models_{\alpha'} \psi$.

Question:

How much space
does it take?

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

- If $\phi(x_1, \dots, x_n) = E(x_i, x_j)$:
answer YES iff $(\alpha(x_i), \alpha(x_j)) \in E$
- If $\phi(x_1, \dots, x_n) = \psi(x_1, \dots, x_n) \wedge \psi'(x_1, \dots, x_n)$:
answer YES iff $G \models_{\alpha} \psi$ and $G \models_{\alpha} \psi'$
- If $\phi(x_1, \dots, x_n) = \neg \psi(x_1, \dots, x_n)$:
answer NO iff $G \models_{\alpha} \psi$
- If $\phi(x_1, \dots, x_n) = \exists y . \psi(x_1, \dots, x_n, y)$:
answer YES iff for some $v \in V$ and $\alpha' = \alpha \cup \{y \mapsto v\}$
we have $G \models_{\alpha'} \psi$.

use 4 pointers \rightsquigarrow LOGSPACE

Question:

How much space
does it take?

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

- If $\phi(x_1, \dots, x_n) = E(x_i, x_j)$:
answer YES iff $(\alpha(x_i), \alpha(x_j)) \in E$
use 4 pointers \rightsquigarrow LOGSPACE
- If $\phi(x_1, \dots, x_n) = \psi(x_1, \dots, x_n) \wedge \psi'(x_1, \dots, x_n)$:
answer YES iff $G \models_{\alpha} \psi$ and $G \models_{\alpha} \psi'$
 $\rightsquigarrow \text{MAX}(\text{SPACE}(G \models_{\alpha} \psi), \text{SPACE}(G \models_{\alpha} \psi'))$
- If $\phi(x_1, \dots, x_n) = \neg \psi(x_1, \dots, x_n)$:
answer NO iff $G \models_{\alpha} \psi$
- If $\phi(x_1, \dots, x_n) = \exists y . \psi(x_1, \dots, x_n, y)$:
answer YES iff for some $v \in V$ and $\alpha' = \alpha \cup \{y \mapsto v\}$
we have $G \models_{\alpha'} \psi$.

Question:

How much space
does it take?

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

- If $\phi(x_1, \dots, x_n) = E(x_i, x_j)$:
answer YES iff $(\alpha(x_i), \alpha(x_j)) \in E$
use 4 pointers \rightsquigarrow LOGSPACE
- If $\phi(x_1, \dots, x_n) = \psi(x_1, \dots, x_n) \wedge \psi'(x_1, \dots, x_n)$:
answer YES iff $G \models_{\alpha} \psi$ and $G \models_{\alpha} \psi'$
 $\rightsquigarrow \text{MAX}(\text{SPACE}(G \models_{\alpha} \psi), \text{SPACE}(G \models_{\alpha} \psi'))$
- If $\phi(x_1, \dots, x_n) = \neg \psi(x_1, \dots, x_n)$:
answer NO iff $G \models_{\alpha} \psi$
 $\rightsquigarrow \text{SPACE}(G \models_{\alpha} \psi)$
- If $\phi(x_1, \dots, x_n) = \exists y . \psi(x_1, \dots, x_n, y)$:
answer YES iff for some $v \in V$ and $\alpha' = \alpha \cup \{y \mapsto v\}$
we have $G \models_{\alpha'} \psi$.

Question:

How much space
does it take?

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

- If $\phi(x_1, \dots, x_n) = E(x_i, x_j)$:
answer YES iff $(\alpha(x_i), \alpha(x_j)) \in E$
use 4 pointers \rightsquigarrow LOGSPACE
- If $\phi(x_1, \dots, x_n) = \psi(x_1, \dots, x_n) \wedge \psi'(x_1, \dots, x_n)$:
answer YES iff $G \models_{\alpha} \psi$ and $G \models_{\alpha} \psi'$
 $\rightsquigarrow \text{MAX}(\text{SPACE}(G \models_{\alpha} \psi), \text{SPACE}(G \models_{\alpha} \psi'))$
- If $\phi(x_1, \dots, x_n) = \neg \psi(x_1, \dots, x_n)$:
answer NO iff $G \models_{\alpha} \psi$
 $\rightsquigarrow \text{SPACE}(G \models_{\alpha} \psi)$
- If $\phi(x_1, \dots, x_n) = \exists y . \psi(x_1, \dots, x_n, y)$:
answer YES iff for some $v \in V$ and $\alpha' = \alpha \cup \{y \mapsto v\}$
we have $G \models_{\alpha'} \psi$.
 $\rightsquigarrow 2 \cdot \log(|G|) + \text{SPACE}(G \models_{\alpha'} \psi)$

Question:

How much space
does it take?

Evaluation problem for FO

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

- If $\phi(x_1, \dots, x_n) = E(x_i, x_j)$:
answer YES iff $(\alpha(x_i), \alpha(x_j)) \in E$ use 4 pointers \rightsquigarrow LOGSPACE
- If $\phi(x_1, \dots, x_n) = \psi(x_1, \dots, x_n) \wedge \psi'(x_1, \dots, x_n)$:
answer YES iff $G \models_{\alpha} \psi$ and $G \models_{\alpha} \psi'$ $\rightsquigarrow \text{MAX}(\text{SPACE}(G \models_{\alpha} \psi), \text{SPACE}(G \models_{\alpha} \psi'))$
- If $\phi(x_1, \dots, x_n) = \neg \psi(x_1, \dots, x_n)$:
answer NO iff $G \models_{\alpha} \psi$ $\rightsquigarrow \text{SPACE}(G \models_{\alpha} \psi)$
- If $\phi(x_1, \dots, x_n) = \exists y . \psi(x_1, \dots, x_n, y)$:
answer YES iff for some $v \in V$ and $\alpha' = \alpha \cup \{y \mapsto v\}$ we have $G \models_{\alpha'} \psi$. $\rightsquigarrow 2 \cdot \log(|G|) + \text{SPACE}(G \models_{\alpha'} \psi)$

Question:

How much space
does it take?

$2 \cdot \log(|G|) + \dots + 2 \cdot \log(|G|) + k \cdot \log(|\alpha| + |G|)$ space
 $\leq |\phi|$ times

Evaluation problem for FO in PSPACE

Input: $\left(\begin{array}{l} \phi(x_1, \dots, x_n) \\ G = (V, E) \\ \alpha = \{x_1, \dots, x_n\} \longrightarrow V \end{array} \right.$

Output: $G \models_{\alpha} \phi ?$

- If $\phi(x_1, \dots, x_n) = E(x_i, x_j)$:
answer YES iff $(\alpha(x_i), \alpha(x_j)) \in E$ use 4 pointers \rightsquigarrow LOGSPACE
- If $\phi(x_1, \dots, x_n) = \psi(x_1, \dots, x_n) \wedge \psi'(x_1, \dots, x_n)$:
answer YES iff $G \models_{\alpha} \psi$ and $G \models_{\alpha} \psi'$ \rightsquigarrow $\text{MAX}(\text{SPACE}(G \models_{\alpha} \psi), \text{SPACE}(G \models_{\alpha} \psi'))$
- If $\phi(x_1, \dots, x_n) = \neg \psi(x_1, \dots, x_n)$:
answer NO iff $G \models_{\alpha} \psi$ \rightsquigarrow $\text{SPACE}(G \models_{\alpha} \psi)$
- If $\phi(x_1, \dots, x_n) = \exists y . \psi(x_1, \dots, x_n, y)$:
answer YES iff for some $v \in V$ and $\alpha' = \alpha \cup \{y \mapsto v\}$ we have $G \models_{\alpha'} \psi$ \rightsquigarrow $2 \cdot \log(|G|) + \text{SPACE}(G \models_{\alpha'} \psi)$

Question:

How much space
does it take?

$2 \cdot \log(|G|) + \dots + 2 \cdot \log(|G|) + k \cdot \log(|\alpha| + |G|)$ space
 $\leq |\phi|$ times

Problem: Usual scenario in database

A **database** of size 10^6

A **query** of size 100

Input:

Problem: Usual scenario in database

A **database** of size 10^6

A **query** of size 100

Input: • query +

Problem: Usual scen

Input: • query +



database

Problem: Usual scenario

Input: • query +

database

But we don't distinguish this in the analysis:

$$\begin{aligned} & \text{TIME}(2^{|query|} + |data|) \\ &= \\ & \text{TIME}(|query| + 2^{|data|}) \end{aligned}$$



Query and data play very **different** roles.

Separation of concerns: How the resources grow with respect to

- the size of the data
- the query size

Combined, Query, and Data complexities

Combined complexity: input size is $|query| + |data|$

Query complexity ($|data|$ fixed): input size is $|query|$

Data complexity ($|query|$ fixed): input size is $|data|$

Combined, Query, and Data complexities

Combined complexity: input size is $|query| + |data|$

Query complexity ($|data|$ fixed): input size is $|query|$

Data complexity ($|query|$ fixed): input size is $|data|$

$O(2^{|query|} + |data|)$ is

exponential in **combined** complexity
exponential in **query** complexity
linear in **data** complexity

$O(|query| + 2^{|data|})$ is

exponential in **combined** complexity
linear in **query** complexity
exponential in **data** complexity

Question

What is the data, query and combined complexity for the evaluation problem for FO?

Remember: **data** complexity, input size: $|data|$
 query complexity, input size: $|query|$
 combined complexity, input size: $|data| + |query|$

$$|\phi| \cdot 2 \cdot \log(|G|) + k \cdot \log(|\alpha| + |G|) \text{ space}$$

Question

What is the data, query and combined complexity for the evaluation problem for FO?

Remember: **data** complexity, input size: $|data|$

query complexity, input size: $|query|$

combined complexity, input size: $|data| + |query|$

$$|\phi| \cdot 2 \cdot \log(|G|) + k \cdot \log(|\alpha| + |G|) \text{ space}$$

The diagram shows the formula $|\phi| \cdot 2 \cdot \log(|G|) + k \cdot \log(|\alpha| + |G|)$ with arrows indicating its components: $|\phi|$ points to "query", $\log(|G|)$ points to "data", k points to "data", and $\log(|\alpha| + |G|)$ points to "data".

$O(\log(|data|) \cdot |query|)$ space

PSPACE combined and query complexity
LOGSPACE data complexity

Evaluation pb for FO is PSPACE-complete

(combined
complexity)

PSPACE-complete problem: **QBF**

(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

Evaluation pb for FO is PSPACE-complete

(combined
complexity)

PSPACE-complete problem: **QBF**

(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

$$\exists p \forall q . (p \vee \neg q) \quad \text{where } p, q \text{ range over } \{T, F\}$$

Evaluation pb for FO is PSPACE-complete

(combined
complexity)

PSPACE-complete problem: **QBF**

(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

$$\exists p \forall q . (p \vee \neg q) \quad \text{where } p, q \text{ range over } \{T, F\}$$

Theorem: Evaluation for FO is PSPACE-complete (combined c.)

Evaluation pb for FO is PSPACE-complete

(combined complexity)

PSPACE-complete problem: **QBF**

(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

$$\exists p \forall q . (p \vee \neg q) \quad \text{where } p, q \text{ range over } \{T, F\}$$

Theorem: Evaluation for FO is PSPACE-complete (combined c.)

Polynomial reduction **QBF** \leadsto **FO** :

1. Given $\psi \in \text{QBF}$,
let $\psi'(x)$ be the replacement
of each 'p' with 'p=x' in ψ .
2. Note: $\exists x \psi'$ holds in a 2-element
graph iff ψ is QBF-satisfiable
3. Test if $G \models \psi'$ for $G = (\{v, v'\}, \{\})$

Evaluation pb for FO is PSPACE-complete

(combined complexity)

PSPACE-complete problem: **QBF**

(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

$$\exists p \forall q . (p \vee \neg q) \quad \text{where } p, q \text{ range over } \{T, F\}$$

Theorem: Evaluation for FO is PSPACE-complete (combined c.)

Polynomial reduction **QBF** \leadsto FO :

$$\psi'(x) = \exists p \forall q . ((p=x) \vee \neg(q=x))$$

1. Given $\psi \in \text{QBF}$,
let $\psi'(x)$ be the replacement of each 'p' with 'p=x' in ψ .
2. Note: $\exists x \psi'$ holds in a 2-element graph iff ψ is QBF-satisfiable
3. Test if $G \models \psi'$ for $G = (\{v, v'\}, \{\})$

Evaluation pb for FO is PSPACE-complete

(combined complexity)

PSPACE-complete problem: **QBF**

(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

$$\exists p \forall q . (p \vee \neg q) \quad \text{where } p, q \text{ range over } \{T, F\}$$

Theorem: Evaluation for FO is PSPACE-complete (combined c.)

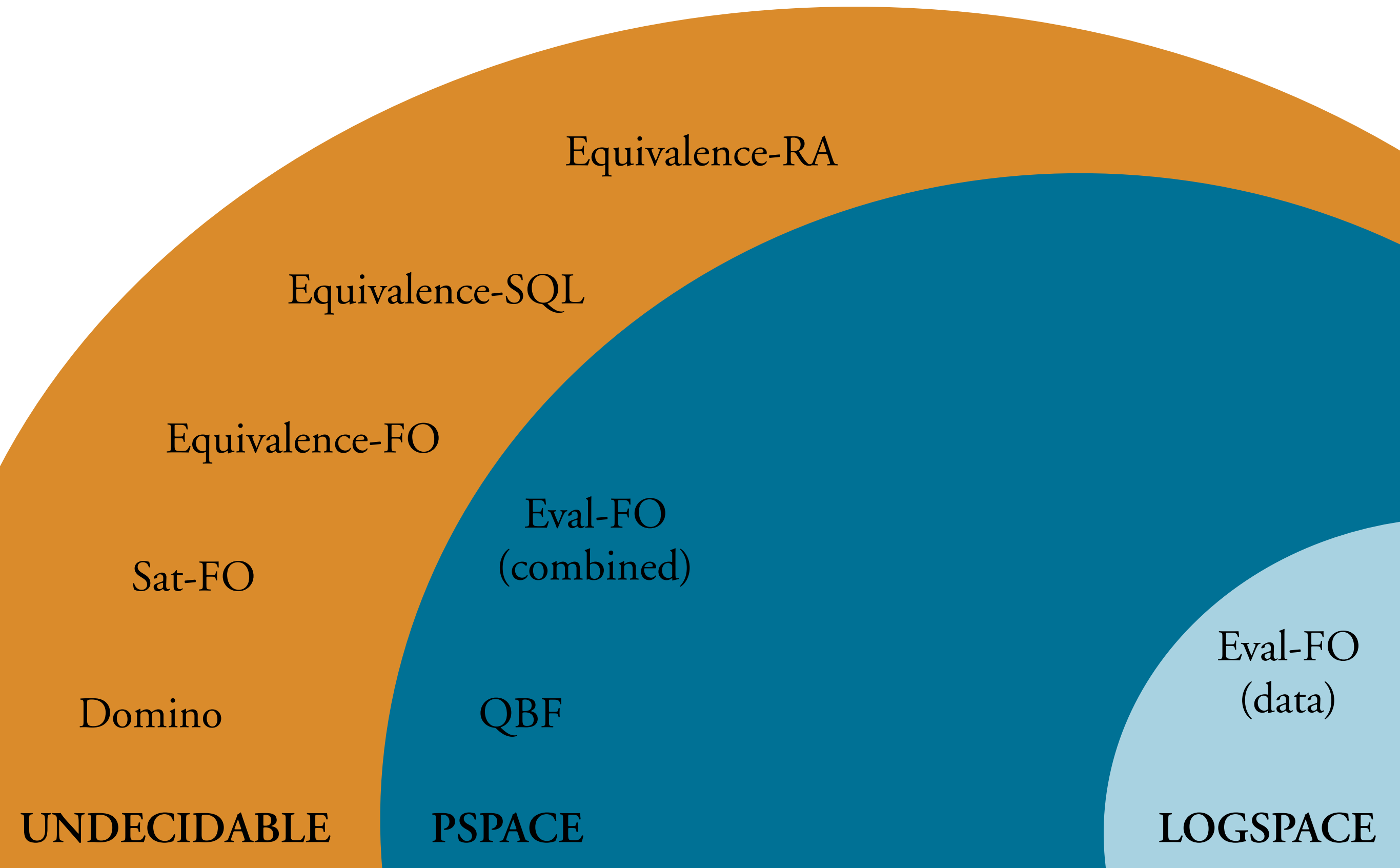
Polynomial reduction **QBF** \leadsto FO :

$$\psi'(x) = \exists p \forall q . ((p=x) \vee \neg(q=x))$$

$$\exists x \exists p \forall q . ((p=x) \vee \neg(q=x))$$

1. Given $\psi \in \text{QBF}$,
let $\psi'(x)$ be the replacement of each 'p' with 'p=x' in ψ .
2. Note: $\exists x \psi'$ holds in a 2-element graph iff ψ is QBF-satisfiable
3. Test if $G \models \psi'$ for $G = (\{v, v'\}, \{\})$

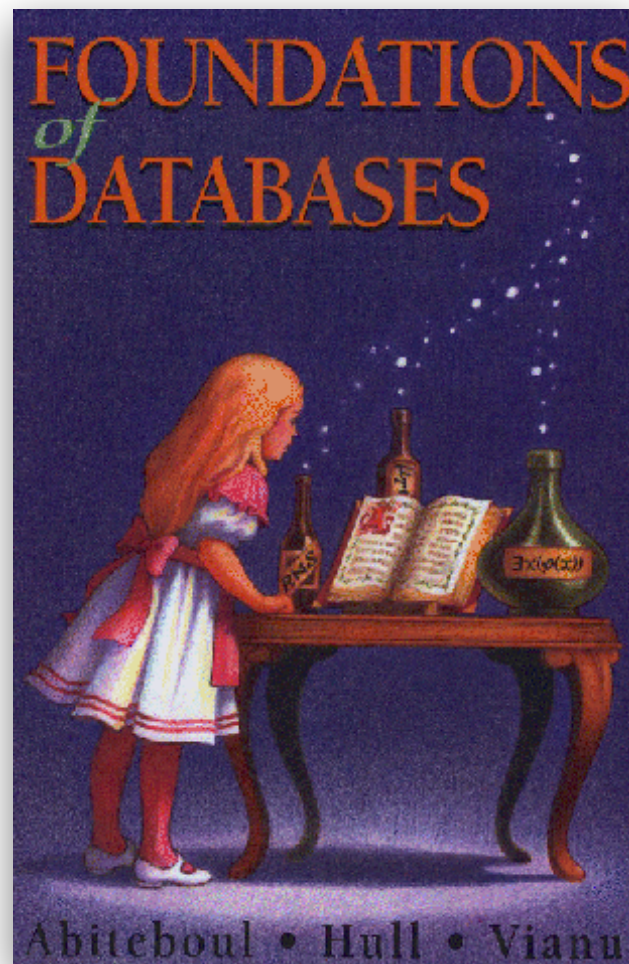
Recap



Bibliography

Abiteboul, Hull, Vianu, “Foundations of Databases”, Addison-Wesley, 1995.

(freely available at <http://webdam.inria.fr/Alice/>)



Chapters 1, 2, 3