Database Theory

Unit 6 — Complexity

Matthias Lanzinger, WS2024/25

TU WIEN Informatics

Complexity of Query Answering

For any query language $\mathscr{L}$ we have the following core decision problem.

$\mathscr{L}$-Eval

Input: a query $q \in \mathscr{L}$, database D

Output: is $q(D) \neq \emptyset$?

Recall, the complexity of $\mathscr{L}$-Eval is what we previously referred to as combined complexity.

Complexity of Query Answering

Additionally, we can study the problem
for each fixed query $q \in \mathscr{L}$.

$\mathscr{L}$**-Eval**$_q$

Input: database D

Output: is $q(D) \neq \emptyset$?

In data complexity:

- $\mathscr{L}$**-Eval** is in complexity class $\mathscr{C}$, if $\mathscr{L}$**-Eval**$_q \in \mathscr{C}$ for every q
- $\mathscr{L}$**-Eval** is hard for $\mathscr{C}$ if $\mathscr{L}$**-Eval**$_q$ is $\mathscr{C}$-hard for some q.

The Story So Far

	Data Complexity	Combined Complexity
First-Order Queries / Relational Algebra	?	?
Conjunctive Queries	?	**NP-** complete
Datalog	**PTIME-** complete	**EXPTIME-** complete

Complexity of Query Answering

It would be just as natural to study the problem for a fixed database D.

$\mathscr{L}$**-Eval**^{D}

Input: query $q \in \mathscr{L}$

Output: is $q(D) \neq \emptyset$?

In query complexity:

- $\mathscr{L}$**-Eval** is in complexity class $\mathscr{C}$, if $\mathscr{L}$**-Eval**$^D \in \mathscr{C}$ for every D
- $\mathscr{L}$**-Eval** is hard for $\mathscr{C}$ if $\mathscr{L}$**-Eval**^{D} is $\mathscr{C}$-hard for some D.

The Story So Far

	Data Complexity	Combined Complexity	Query Complexity
First-Order Queries / Relational Algebra	?	?	?
Conjunctive Queries	?	**NP-** complete	?
Datalog	**PTIME-** complete	**EXPTIME-** complete	?

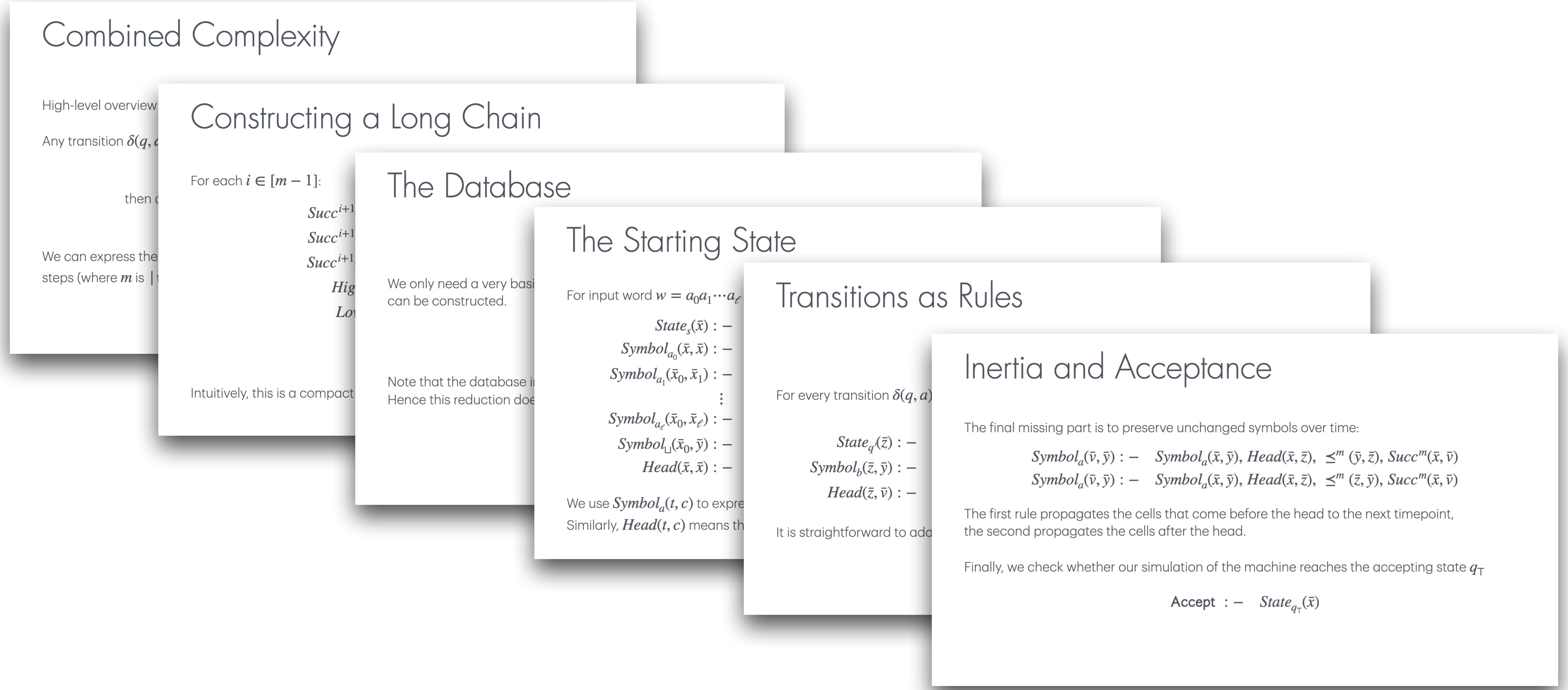
Datalog — Program Complexity

Recall, a Datalog query is a tuple (Π, q) consisting of program and an atomic query.
Hence, query complexity is also referred to as program complexity in this context.

What is the program complexity of Datalog?

Datalog — Program Complexity

Recall our **EXPTIME**-hardness proof for combined complexity.
We constructed a program that simulates an **EXPTIME** Turing Machine.



Datalog — Program Complexity

Recall our **EXPTIME**-hardness proof for combined complexity.
We constructed a program that simulates an **EXPTIME** Turing Machine.

The Database

We only need a very basic database from which our successor relationship can be constructed.

$$D = \{\ Succ^1(0{,}1),\ High^1(1),\ Low^1(0)\ \}$$

Note that the database in this construction is *independent* of the input w!
Hence this reduction does not work to establish the complexity in data complexity.

Datalog — Program Complexity

Recall our **EXPTIME**-hardness proof for combined complexity.
We constructed a program that simulates an **EXPTIME** Turing Machine.

The Database

We only need a very basic database from which our successor relationship can be constructed.

$$D = \{\ Succ^1(0,1),\ High^1(1),\ Low^1(0)\ \}$$

Note that the database in this construction is *independent* of the input w!
Hence this reduction does not work to establish the complexity in data complexity.

We used the same database D for all input programs!

The same reduction also shows that ***Datalog-Eval***^{D} is **EXPTIME**-hard.

Thus, Datalog-EVAL is **EXPTIME**-hard in query complexity.

Datalog — Program Complexity

Recall our **EXPTIME**-hardness proof for combined complexity.
We constructed a program that simulates an **EXPTIME** Turing Machine.

The Database

We only need a very basic database from which our successor relationship can be constructed.

$$D = \{\ Succ^1(0,1),\ High^1(1),\ Low^1(0)\ \}$$

Note that the database in this construction is *independent* of the input w!
Hence this reduction does not work to establish the complexity in data complexity.

The upper bound is trivially inherited from combined complexity:

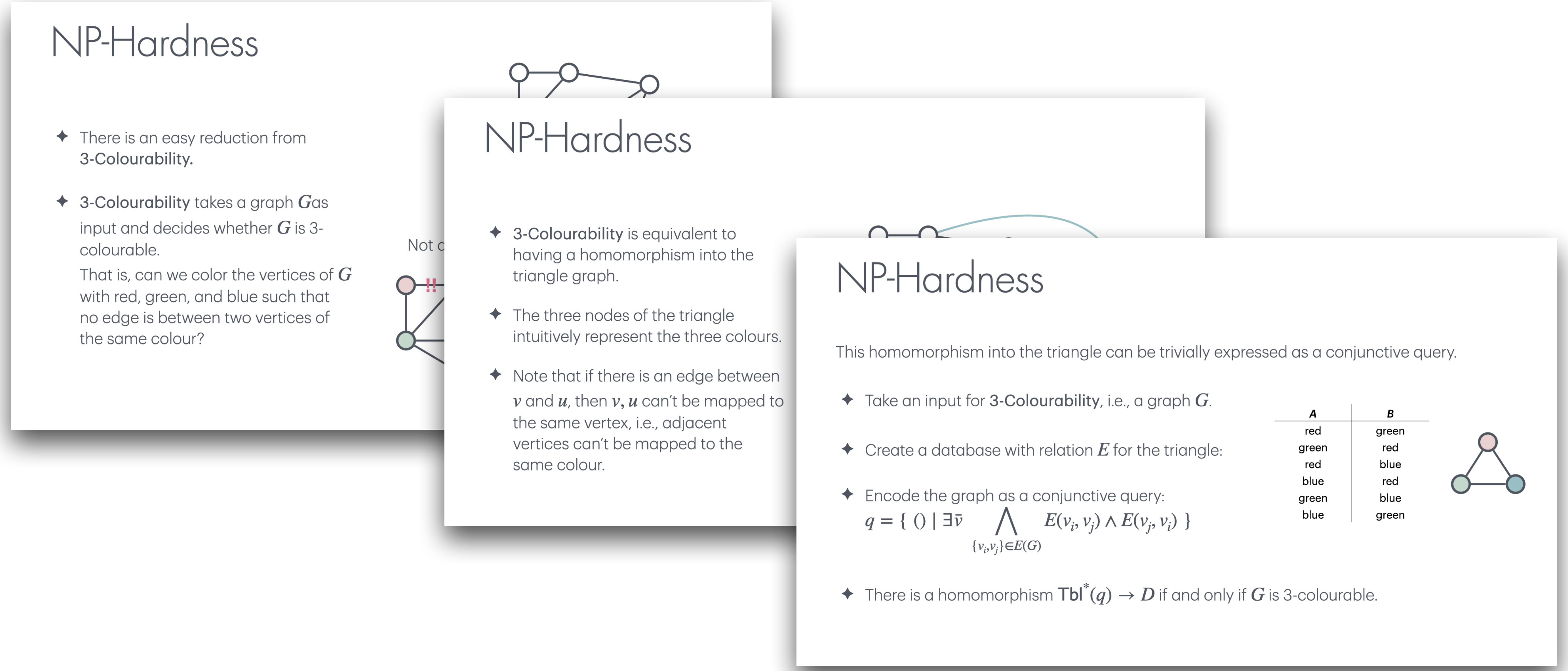
If there is an **EXPTIME** algorithm for any combination of q and D, then there the algorithm will also be in **EXPTIME** for a fixed D.

Filling the Table

	Data Complexity	Combined Complexity	Query Complexity
First-Order Queries / Relational Algebra	?	?	?
Conjunctive Queries	?	**NP-** complete	?
Datalog	**PTIME-** complete	**EXPTIME-** complete	**EXPTIME-** complete

CQ Query Complexity

Again it is enough to recall the reduction
we used to establish combined complexity.



CQ Query Complexity

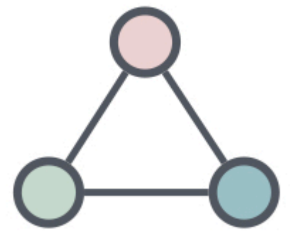
Again it is enough to recall the reduction
we used to establish combined complexity.

NP-Hardness

This homomorphism into the triangle can be trivially expressed as a conjunctive query

- Take an input for **3-Colourability**, i.e., a graph G.
- Create a database with relation E for the triangle:
- Encode the graph as a conjunctive query:
  $q = \{\ ()\mid \exists \bar{v} \bigwedge_{\{v_i,v_j\}\in E(G)} E(v_i, v_j) \wedge E(v_j, v_i)\ \}$

A	*B*
red	green
green	red
red	blue
blue	red
green	blue
blue	green



- There is a homomorphism $\mathsf{Tbl}^*(q) \to D$ if and only if G is 3-colourable.

Filling the Table

	Data Complexity	Combined Complexity	Query Complexity
First-Order Queries / Relational Algebra	?	?	?
Conjunctive Queries	?	**NP**-complete	**NP**-complete
Datalog	**PTIME**-complete	**EXPTIME**-complete	**EXPTIME**-complete

First-Order Queries

FO Queries

- Without loss of generality, every FO query is of the form

$$q = \{ \ \bar{z} \mid \ \exists x_1 \forall y_1 \cdots \exists x_n \forall y_n \ \varphi(x_1, y_1, \ldots, x_n, y_n) \ \}$$

- Let us use $Adom = \{a_1, \ldots, a_m\}$ for the elements in the *active domain* of q and D.

- We will define two procedures **eval**$_\exists$ and **eval**$_\forall$ that call each other recursively, and access the same global variables $X = \{x_1, y_1, \ldots, x_n, y_n\}$. Calling **eval**$_\exists$ on the formula of q will return **true** if and only if $q(D) \neq \emptyset$.

FO-Eval Algorithm

func **eval**$_\exists(i)$
  for $x_i \in Adom$
    if **eval**$_\forall(i)$ returns **true**
    then return **true**
  return **false**

The proof for why this is correct should be clear. We are just fully enumerating all possible assignments and testing all of them.

func **eval**$_\forall(i)$
  for $y_i \in Adom$
    if $i = n$
      if φ evaluates to false under current assignment to $x_1, y_1, \ldots, x_n, y_n$
      then return **false**
    else
      if **eval**$_\exists(i+1)$ returns **false**
      then return **false**
  return **true**

How much space does it require to run this algorithm?

FO-Eval Algorithm

Non-trivial parts that take up space:

- **The global variables X:**

  There are $2n$ variables (x_i, y_i for every $1 \leq i \leq n$).

  Each of them stores elements from $Adom \rightarrow \log|Adom|$ space per variable.

  $= O(n \log|Adom|)$ bits to store X

- **The stack for the recursion:**

  The recursion depth is at most $2n$.

  At every step of the recursion, we need to remember

  a pointer for where to return to, and the argument i.

  Since $i \leq n$, we need $O(n \log(n))$ bits to store the recursion stack.

FO-Eval Algorithm

Non-trivial parts that take up space:

- Evaluation of φ for fixed assignment:

  Requires a traversal of the syntax-tree of φ and lookups into the database.

  $O(\log|\varphi| + \log|D|)$ space suffices to do this.

FO-Eval Algorithm

In total we need space in the order of

$$O(n \log n + n \log |Adom| + \log |\varphi| + \log |D|)$$

Depends on query q
(n, φ, and $Adom$)

Depends on database D
($Adom$ and D)

FO-Eval Algorithm

In total we need space in the order of

$$O(n \log n + n \log |Adom| + \log |\varphi| + \log |D|)$$

Depends on query q
(n, φ, and $Adom$)

Depends on database D
($Adom$ and D)

$$O(n \log n + n \log |Adom| + \log |\varphi|)$$

$$O(\log |Adom| + \log |D|)$$

FO-Eval Algorithm

In total we need space in the order of

$$O(n \log n + n \log |Adom| + \log |\varphi| + \log |D|)$$

Theorem
FO-Eval is in **PSPACE** in combined complexity.
FO-Eval is in **L** (log space) in data complexity.
FO-Eval is in **PSPACE** in query complexity.

FO-Eval Complexity

The quantified SAT problem (***QSAT***) is **PSPACE**-hard.

We can reduce ***QSAT*** to ***FO-Eval*** with a fixed database.
(we leave this as an easy but interesting exercise)

→ ***FO-Eval*** is **PSPACE**-complete in both combined and query complexity

QSAT

Input instances of the form

$\Phi = Q_1 x_1 \, Q_2 x_2 \, \cdots \, Q_n x_n \, \psi(x_1, x_2, \ldots, x_n)$

where:

- $Q_i \in \{\forall, \exists\}$
- $x_1, x_2, \ldots, x_n$ are Boolean variables (can be either **true** or **false**)
- $\psi(x_1, x_2, \ldots, x_n)$ is a propositional formula

Example

$\forall x_1, x_4 \, \exists x_2, x_3 \, (x_1 \vee \neg x_2) \wedge (x_3 \vee x_4) \wedge (\neg x_1 \vee x_3)$

Filling the Table

	Data Complexity	Combined Complexity	Query Complexity
First-Order Queries / Relational Algebra	in **L**	**PSPACE**-complete	**PSPACE**-complete
Conjunctive Queries	?	**NP**-complete	**NP**-complete
Datalog	**PTIME**-complete	**EXPTIME**-complete	**EXPTIME**-complete

Filling the Table

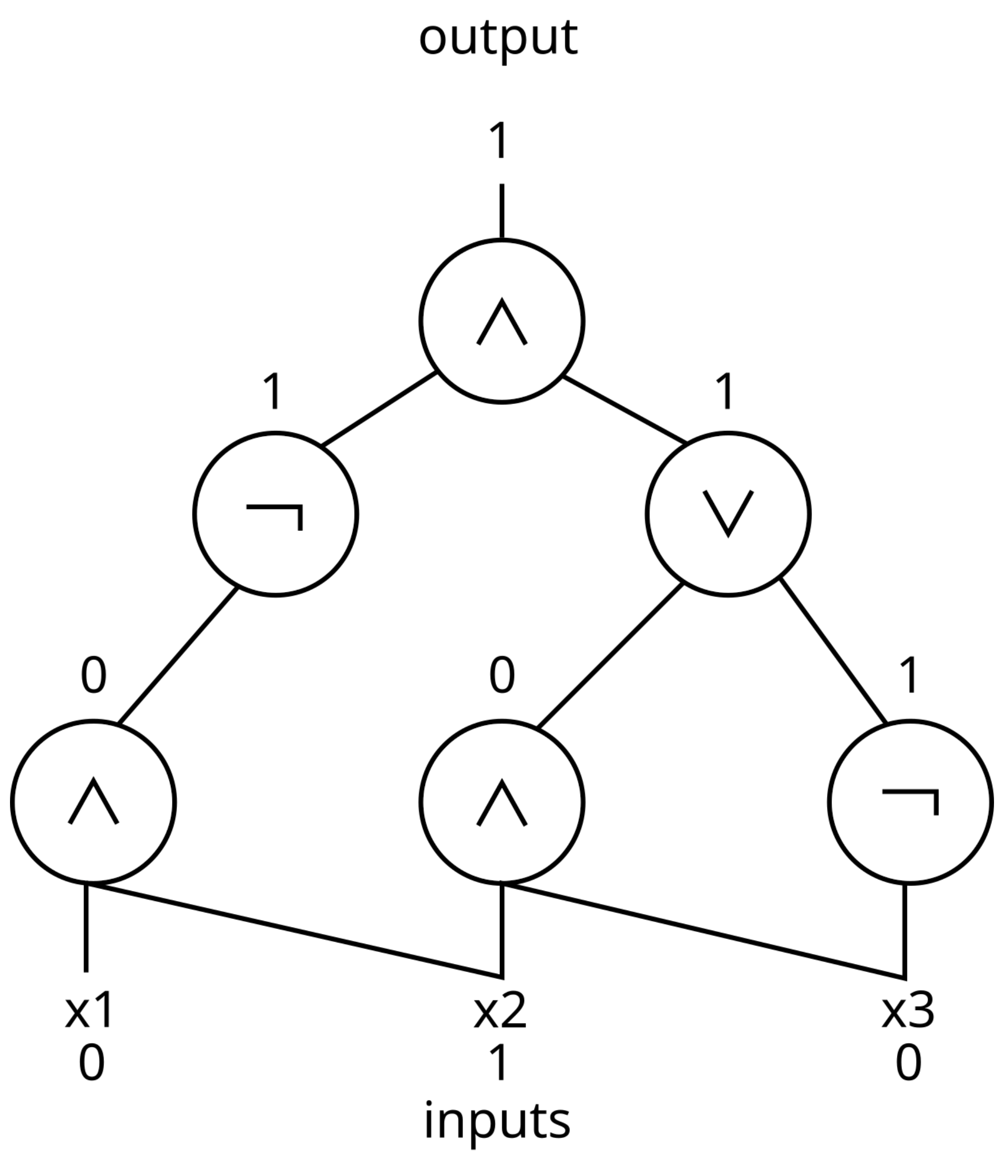
CQs are a special case FO-queries. Our algorithm from before works also for them!

	Data Complexity	Combined Complexity	Query Complexity
First-Order Queries / Relational Algebra	in **L**	**PSPACE**-complete	**PSPACE**-complete
Conjunctive Queries	in **L**	**NP**-complete	**NP**-complete
Datalog	**PTIME**-complete	**EXPTIME**-complete	**EXPTIME**-complete

Data Complexity

Logarithmic space (**L**) is great, but its not the best we can do.

Complexity below **L** will require a different perspective on complexity: Boolean Circuits.



Source: https://en.wikipedia.org/wiki/Circuit_complexity

Boolean Circuits

A directed acyclic graph, 2 kinds of nodes:

- Inputs (nodes with no in-edge)
- Gates (AND, OR, NOT)

The fan-in of a gate is the number of ingoing edges. NOT gates always have fan-in 1.

There is exactly one node with no out-edges. We call it the output gate.

Boolean Circuits

We can now define complexity classes by problems that can be decided by different kinds of circuits. That is, the circuit outputs 1 on every “yes”-instance, and 0 otherwise. Circuit size usually depends on the size n of the input word (the number of input bits).

- NC^i is the class of problems decided by a circuit with $O(\log^i(n))$ depth and a polynomial number of gates with fan-in at most 2
- AC^i is the class of problems decided by a circuit with $O(\log^i(n))$ depth and a polynomial number of gates with unbounded fan-in

Circuit Complexity

Circuit complexity are interesting for parallelizability and related questions:

For example, a problem in NC^i can be solved in $O(\log^i(n))$ time using polynomial many processors.
Idea: each level of the circuit can evaluate its gates in parallel. With enough processors, we only need matching the depth of the circuit.

$$\underbrace{\mathsf{AC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{NC}^2}_{\text{“highly parallelizable”}} \subseteq \mathsf{AC}^2 \subseteq \mathsf{NC}^3 \cdots \subseteq \mathsf{P}$$

Circuit Complexity

In data complexity, FO-Eval is in AC^0.
Constant depth circuits are enough!

(Very!) Simplified, the idea is that for every quantifier free sub formula $\varphi(x_1, x_2, \ldots, x_n)$ we can create a gate that expresses whether it holds or not for every assignment of $x_1, \ldots, x_n$ to constants $c_1, \ldots, c_n \in Dom$.

Universal quantification $\forall \bar{x} \varphi(\bar{x})$ then corresponds to one large AND gate that takes all gates for every $\varphi(\bar{c})$ as input.

Analogously, existential quantification $\exists \bar{x} \varphi(\bar{x})$ becomes a large OR with the respective gates for $\varphi(\bar{c})$ as input.

Putting AC^0 in Context

Consider the following simple problem

Parity

Input: a string of 1s and 0s.

Output: does the string contain an even number of 1s?

Parity is not in AC^0!

Filling the Table

	Data Complexity	Combined Complexity	Query Complexity
First-Order Queries / Relational Algebra	in **AC0**	**PSPACE-** complete	**PSPACE-** complete
Conjunctive Queries	in **AC0**	**NP-** complete	**NP-** complete
Datalog	**PTIME-** complete	**EXPTIME-** complete	**EXPTIME-** complete

But databases work fine?

They do, until they don't.

Typical applications have converged on using easy queries, where the joins are mostly simple.

However, even on very simple queries, like counting paths in graphs, we clearly see exponential scaling behaviour on modern systems.

Example query path-04:
$|\{\bar{x} \mid E(x_1, x_2) \land E(x_2, x_3) \land E(x_3, x_4) \land E(x_4, x_5)\}|$
i.e., the number of 4-edge paths.

	SparkSQL	**Yannakakis-inspired evaluation**
path-03	6.3s	1.59s
path-04	51s	1.76s
path-05	401s	2.03s
path-06	out of memory	2.18s

From: Lanzinger, M., Pichler, R., & Selzer, A. (2024). Avoiding Materialisation for Guarded Aggregate Queries. *arXiv preprint arXiv:2406.17076*.

Practical Data Complexity

Can the lower complexity classes for data complexity
be useful in practice?

- Difficult to use them in general database systems. These systems require algorithms that work on every input query.

- Potential use-cases in cases where we have specialised hardware/systems for specific queries.
  Recall that $\mathsf{AC}^0 \subseteq \mathsf{NC}^1$, hence with more practical bounded fan-in logarithmic depth circuits suffice. Could be built into hardware.

Descriptive Complexity

(A very imprecise introduction)

Disclaimer

We will focus on the high-level ideas of descriptive complexity.

In the interest of accessibility, we will omit various important technical details. Importantly, domains are always finite in this setting (based on the idea that computation is inherently finite).

These slides are not suitable as a reference on descriptive complexity. For a formally reliable reference please refer to "Immerman, N., 1998. *Descriptive complexity*. Springer."

Descriptive Complexity

- We want to connect computational complexity classes with query languages.
- Say I want to query a property of a database that I know is in complexity class $\mathscr{C}$. Is it possible to do this in language $\mathscr{L}$?

  Examples for a graph database:

  Finding a large clique is in NP: what query language can I use?

  Deciding whether the graph is strongly connected is in NL: what query language can I use?

Descriptive Complexity

Ultimately, what we want is statements of the following form:

For every problem $P \in \mathscr{C}$ there exists a formula φ in language $\mathscr{L}$ such that:

$$I \in P \iff I \models \varphi$$

and for every $\psi \in \mathscr{L}$, the language $\{I \mid I \models \psi\}$ is in $\mathscr{C}$.

As a shorthand for this relationship, we write $\mathscr{C} \equiv \mathscr{L}$.

Note that we implicitly treat all languages here as databases, and assume formulas over the same vocabulary..

$AC^0 \equiv$ FO

We've already seen that for every FO formula φ, there exists an AC^0 circuit C such that:

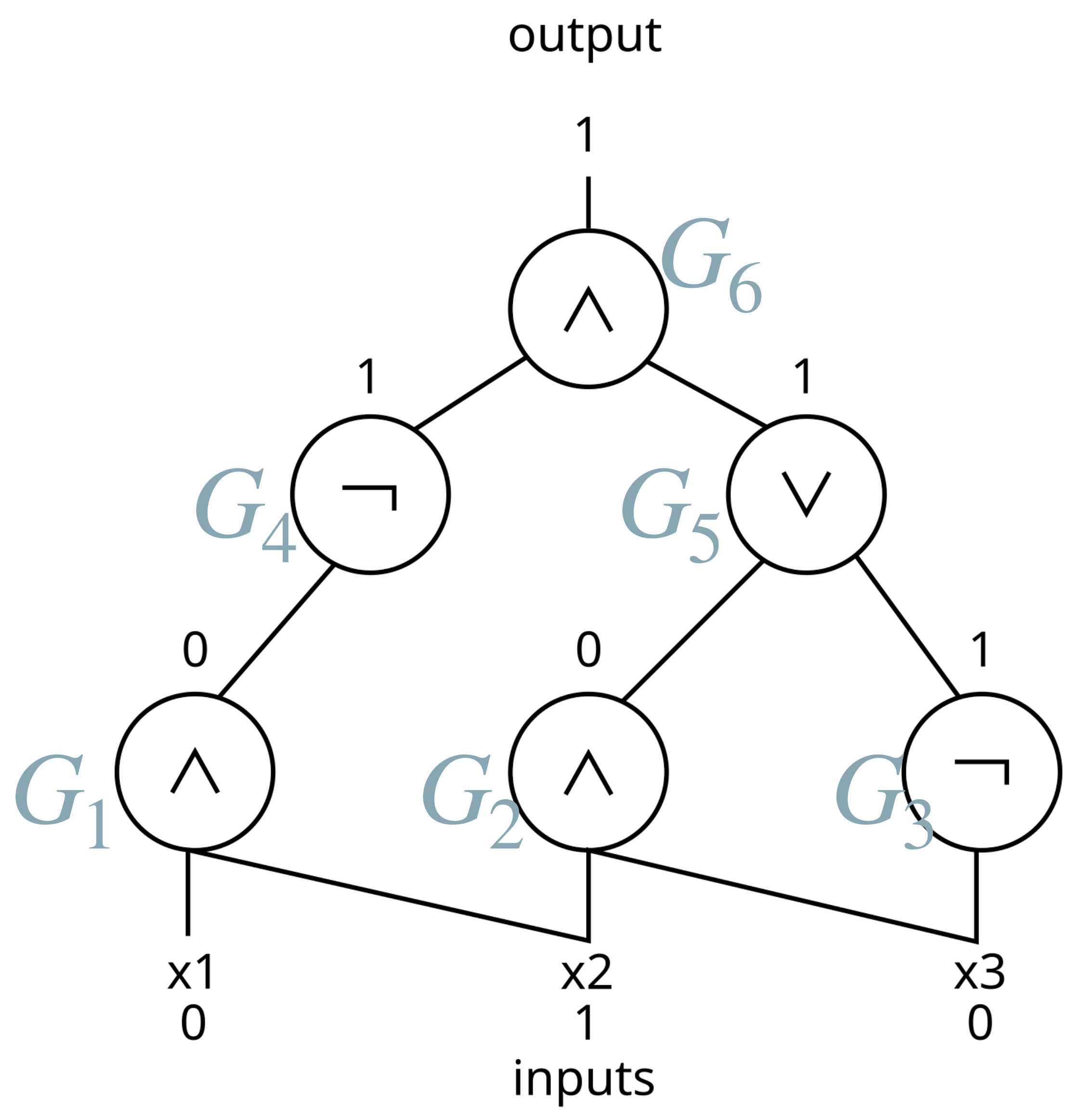
$$C \text{ outputs } 1 \text{ on input } D \Longleftrightarrow D \models \varphi$$

To see that $AC^0 \equiv$FO, we need to also show the opposite direction:

For every circuit C there is an equivalent FO formula.

$AC^0 \equiv FO$

For every circuit C there is an equivalent FO formula:
Unroll the circuit into a formula.



Source: https://en.wikipedia.org/wiki/Circuit_complexity

One relation:

$Input(i, v)$... Input index i has value v

we use the abbreviation $I(x) := Input(x, \text{True})$

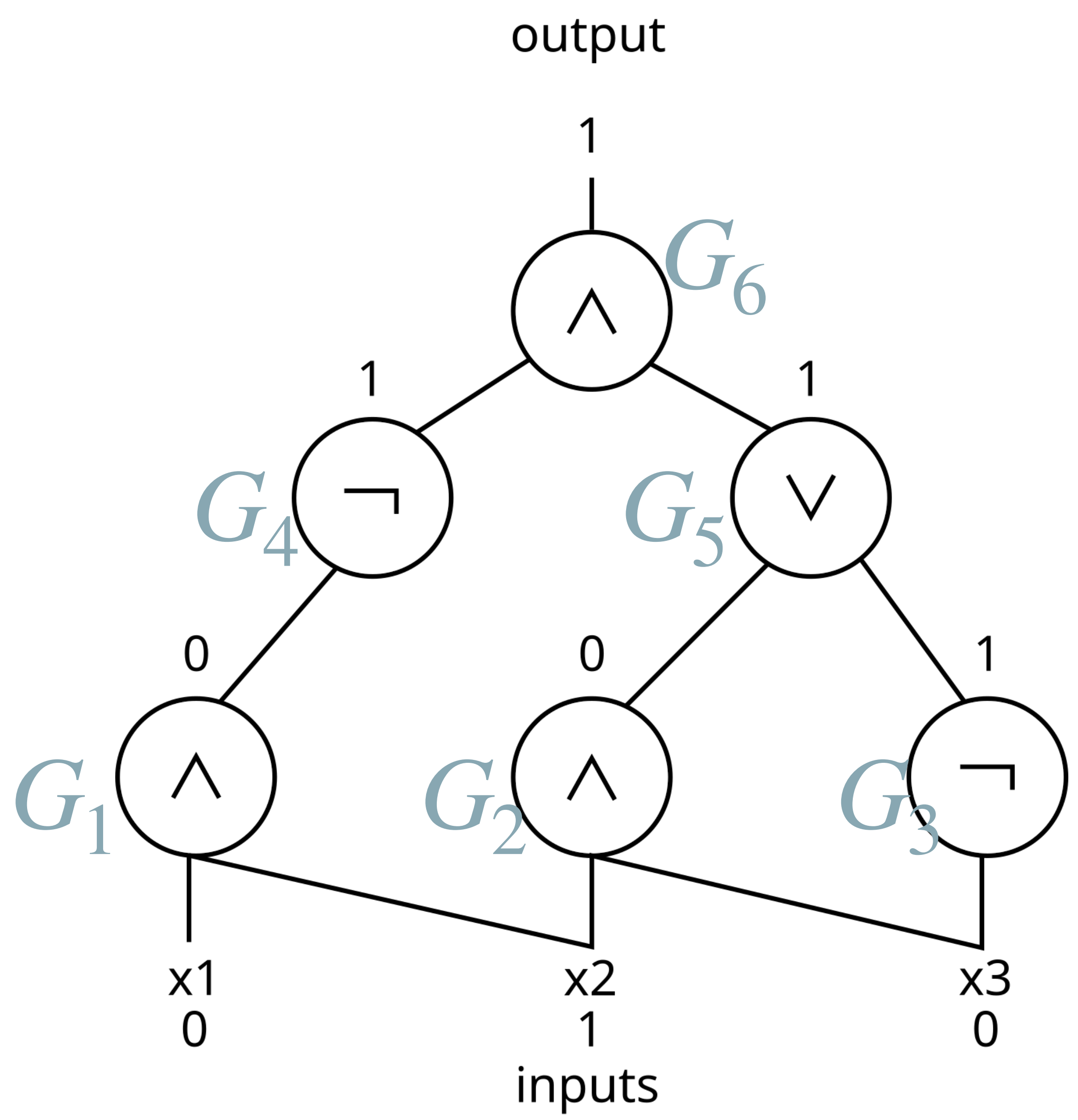
$$G_6 := G_4 \wedge G_5$$

$$G_4 := \neg G_1 \qquad G_5 := G_2 \vee G_3$$

$$G_1 := I(x_1) \wedge I(x_2) \qquad G_2 := I(x_2) \wedge I(x_3) \qquad G_3 := \neg I(x_3)$$

AC^0 = FO

For every circuit C there is an equivalent FO formula:
Unroll the circuit into a formula.



Source: https://en.wikipedia.org/wiki/Circuit_complexity

One relation:

$Input(i, v)$... Input index i has value v

we use the abbreviation $I(x) := Input(x, \text{True})$

$$\varphi = G_6 := \neg(I(x_1) \wedge I(x_2)) \wedge \big((I(x_2) \wedge I(x_3)) \vee \neg I(x_3)\big)$$

Second-Order Logic

Second-Order Logic

- So far we have focused on first-order logic, that is, logic where we can quantify over the objects of the domain:
  "There is some object $a \in Dom$ such that the formula is true if we interpret variable x as a.
- A next natural step is to allow quantification over relations:
  "There is some relation $A \subseteq Dom^k$ such that the formula is true if interpret second-order variable X as A.

Second-Order Formulas

Like first-order formulas but we also allow quantification $\forall X$ and $\exists X$ where X is a *relation variable*.

$$\exists C \overbrace{\Big(\forall x\, C(x) \rightarrow V(x) \Big)}^{C \subseteq V} \wedge |C| \leq k$$

$$\wedge \Big(\underbrace{\forall yz, E(y,z) \rightarrow (C(y) \vee C(z)}_{\text{Every edge has one endpoint in } C} \Big)$$

$\exists SO$

$\exists SO$ is the language of second order formulas where second order quantification is always existential.

$$\textbf{NP} \equiv \exists SO$$

Intuition $\exists SO \subseteq$ **NP:** A second order variable of arity k has at most $|Dom|^k$ tuples. Since k is fixed (part of the query), we can make polynomial guesses for all relations and then simply check the formula for the database extended by the guesses.

$\exists SO$

Intuition $\exists SO \supseteq$ **NP:** Suppose a NTM that takes at most $n^\alpha - 1$ time for problem P. We express this as a formula

$$\exists C_1 \exists C_2 \cdots \exists C_g \exists \Delta \,.\, \varphi(\bar{C}, \Delta)$$

$C_i(\bar{s}, \bar{t})$:= cell $\bar{s}$ at time $\bar{t}$ contains symbol i

Encodes the non-deterministic choices. Assume choices are always binary, then $\Delta(\bar{t})$ intuitively is true iff choice 1 is made.

φ is similar to our reduction of **EXPTIME** TMs to Datalog.

$\forall SO$

$\forall SO$ is the language of second order formulas where second order quantification is always universal.

co-NP $\equiv \forall SO$

Same idea as for $\exists SO$ considering $\forall A\varphi \equiv \neg \exists \neg \varphi$.

Full Second-Order?

$$\textbf{PH} \equiv SO$$

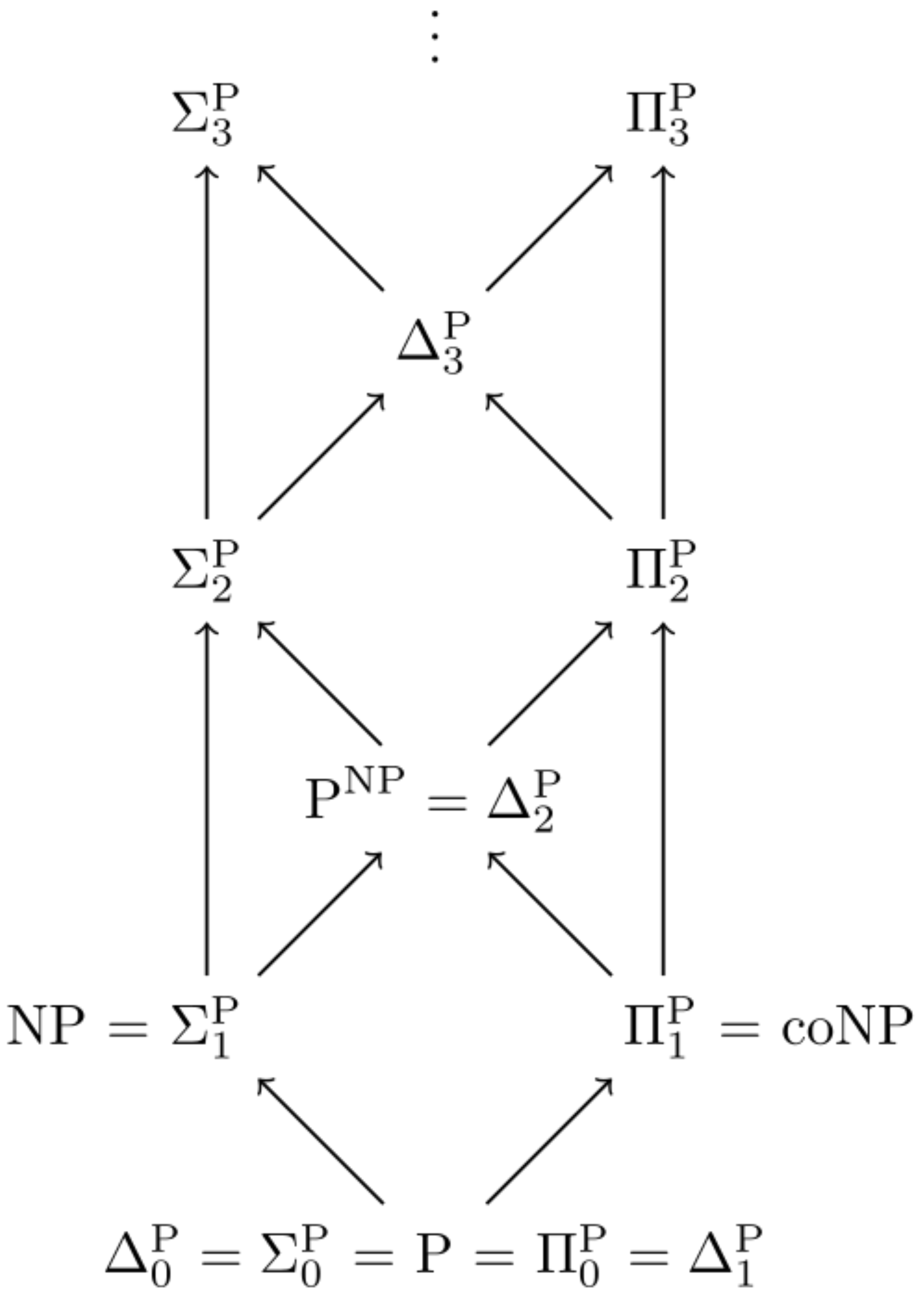
Recall the definition of PH:

$\Sigma^P_{k+1} = \textbf{NP}^{\Sigma^P_k}$ $\qquad \Pi^P_{k+1} = \textbf{co-NP}^{\Sigma^P_k}$

Recall $\mathscr{C}^{\mathscr{O}}$ means in complexity class $\mathscr{C}$ with an $\mathscr{O}$ oracle.

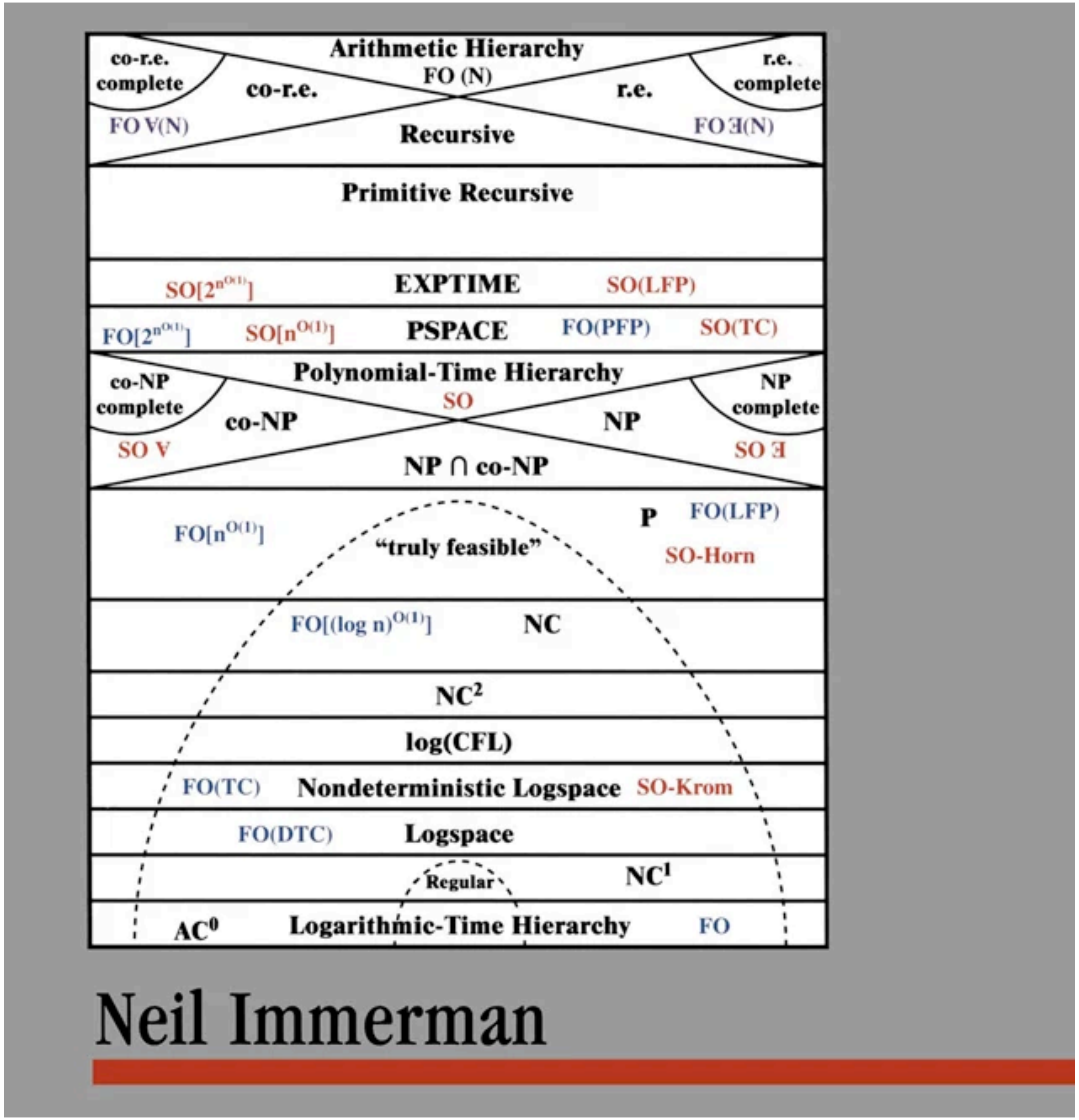
Intuition: a SO formula $\forall A\, \exists B\, \varphi(A, B)$ can be seen as a $\forall SO$ formula $\forall A\, \psi(A)$ where $\psi \in \exists SO$. That is, a **co-NP** problem if we have an $\exists SO = \textbf{NP}$ oracle.

By induction this idea extends through the whole hierarchy.



Source: https://en.wikipedia.org/wiki/Polynomial_hierarchy

There's More



Source:
Immerman, N., 1998.
Descriptive complexity.
Springer.

Summary

- We have a precise idea of how difficult it is to evaluate various query languages.
- We can make more fine-grained observations about the role of database size and query size in the evaluation: query and data complexity.
- The reasoning behind the complexity results reveals important insights for how the theory is connected to practice.
- Descriptive Complexity tightly links expressivity of query languages to computational complexity