TECHNISCHE UNIVERSITÄT DRESDEN

# DATABASE THEORY

## Lecture 5: Complexity of FO Query Answering (II)

**David Carral**

**Knowledge-Based Systems**

TU Dresden, April 21, 2020

# Review: Query Complexity

Query answering as decision problem
$\leadsto$ consider Boolean queries

Various notions of complexity:

- Combined complexity (complexity w.r.t. size of query and database instance)
- Data complexity (worst case complexity for any fixed query)
- Query complexity (worst case complexity for any fixed database instance)

Various common complexity classes:

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSpace} \subseteq \mathrm{ExpTime}$$

# Review: FO Combined Complexity

**Theorem 4.1** The evaluation of FO queries is PSpace-complete with respect to combined complexity.

We have actually shown something stronger:

**Theorem 4.2** The evaluation of FO queries is PSpace-complete with respect to query complexity.

This also holds true when restricting to domain-independent queries.

# Data Complexity of FO Query Answering

The algorithm showed that FO query evaluation is in L
$\leadsto$ can we do any better?

**What could be better than L?**

$$? \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \ldots$$

$\leadsto$ we need to define circuit complexities first

# Boolean Circuits

**Definition 5.1:** A Boolean circuit is a finite, directed, acyclic graph where

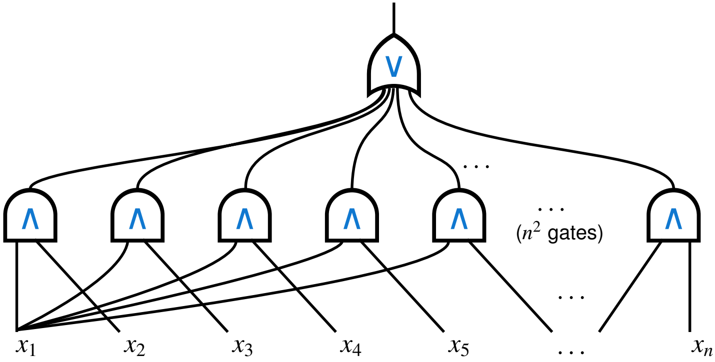
- each node that has no predecessors is an input node
- each node that is not an input node is one of the following types of logical gate: AND, OR, NOT
- one or more nodes are designated output nodes

$\leadsto$ we will only consider Boolean circuits with exactly one output

$\leadsto$ propositional logic formulae are Boolean circuits with one output and gates of fanout $\leq 1$

# Example

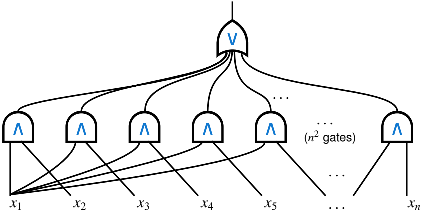
A Boolean circuit over an input string $x_1 x_2 \ldots x_n$ of length $n$




Corresponds to formula $(x_1 \land x_2) \lor (x_1 \land x_3) \lor \ldots \lor (x_{n-1} \land x_n)$
$\leadsto$ accepts all strings with at least two 1s

# Circuits as a Model for Parallel Computation

Previous example:




$\leadsto$ $n^2$ processors working in parallel

$\leadsto$ computation finishes in $2$ steps

- size: number of gates = total number of computing steps
- depth: longest path of gates = time for parallel computation

$\leadsto$ circuits as a refinement of polynomial time that takes parallelizability into account

# Solving Problems With Circuits

**Observation:** the input size is "hard-wired" in circuits
$\leadsto$ each circuit only has a finite number of different inputs
$\leadsto$ not a computationally interesting problem

How can we solve interesting problems with Boolean circuits?

**Definition 5.2:** A uniform family of Boolean circuits is a set of circuits $C_n$ $(n \geq 0)$ that can easily[a] be computed from $n$.

A language $\mathcal{L} \subseteq \{0, 1\}^*$ is decided by a uniform family $(C_n)_{n \geq 0}$ of Boolean circuits if for each word $w$ of length $|w|$:

$$w \in \mathcal{L} \quad \text{if and only if} \quad C_{|w|}(w) = 1$$

[a] We don't discuss the details here; see course Complexity Theory.

# Measuring Complexity with Boolean Circuits

How to measure the computing power of Boolean circuits?

**Relevant metrics:**

- size of the circuit: overall number of gates (as function of input size)
- depth of the circuit: longest path of gates (as function of input size)
- fan in: two inputs per gate or any number of inputs per gate?

Important classes of circuits: small-depth circuits

**Definition 5.3:** $(C_n)_{n\geq 0}$ is a family of small-depth circuits if

- the size of $C_n$ is polynomial in $n$,
- the depth of $C_n$ is poly-logarithmic in $n$, that is, $O(\log^k n)$.

# The Complexity Classes NC and AC

Two important types of small-depth circuits:

**Definition 5.4:** $\mathsf{NC}^k$ is the class of problems that can be solved by uniform families of circuits $(C_n)_{n\geq 0}$ of fan-in $\leq 2$, size polynomial in $n$, and depth in $O(\log^k n)$.

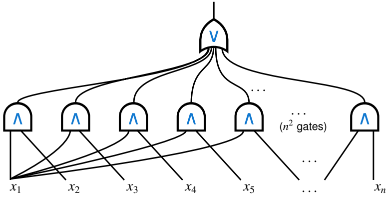
The class NC is defined as $\mathsf{NC} = \bigcup_{k\geq 0} \mathsf{NC}^k$.

("Nick's Class" named after Nicholas Pippenger by Stephen Cook)

**Definition 5.5:** $\mathsf{AC}^k$ and AC are defined like $\mathsf{NC}^k$ and NC, respectively, but for circuits with arbitrary fan-in.

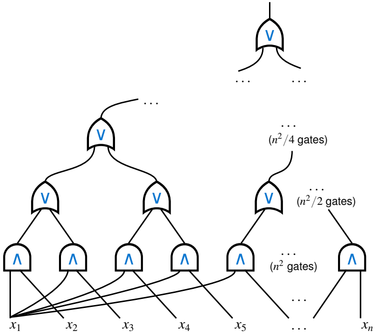
(A is for "Alternating": AND-OR gates alternate in such circuits)

# Example




family of polynomial size,
constant depth,
arbitrary fan-in circuits
$\rightsquigarrow$ in $AC^0$

We can eliminate arbitrary fan-ins by using more layers of gates:




family of polynomial size,
logarithmic depth,
bounded fan-in circuits
$\rightsquigarrow$ in $NC^1$

# Relationships of Circuit Complexity Classes

**The previous sketch can be generalised:**

$$\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}^1 \subseteq \ldots \subseteq \mathsf{AC}^k \subseteq \mathsf{NC}^{k+1} \subseteq \ldots$$

Only few inclusions are known to be proper: $\mathsf{NC}^0 \subset \mathsf{AC}^0 \subset \mathsf{NC}^1$

Direct consequence of above hierarchy: $\mathsf{NC} = \mathsf{AC}$

**Interesting relations to other classes:**

$$\mathsf{NC}^0 \subset \mathsf{AC}^0 \subset \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{AC}^1 \subseteq \ldots \subseteq \mathsf{NC} \subseteq \mathsf{P}$$

**Intuition:**

- Problems in NC are parallelisable (known from definition)
- Problems in P \ NC are inherently sequential (educated guess)

However: it is not known if $\mathsf{NC} \neq \mathsf{P}$

# Back to Databases . . .

**Theorem 5.6:** The evaluation of FO queries is complete for (logtime uniform) $\mathrm{AC}^0$ with respect to data complexity.

**Proof:**

- Membership: For a fixed Boolean FO query, provide a uniform construction for a small-depth circuit based on the size of a database
- Hardness: Show that circuits can be transformed into Boolean FO queries in logarithmic time (not on a standard TM . . . not in this lecture)

# From Query to Circuit

**Assumptions:**

- query and database schema is fixed
- database instance (and thus active domain) are variable

Construct circuit uniformly based on size of active domain

**Sketch of construction:**

- one input node for each possible database tuple (over given schema and active domain)
  $\leadsto$ true or false depending on whether tuple is present or not
- Recursively, for each subformula, introduce a gate for each possible tuple (instantiation) of this formula
  $\leadsto$ true or false depending on whether the subformula holds for this tuple or not
- Logical operators correspond to gate types: basic operators obvious, $\forall$ as generalised conjunction, $\exists$ as generalised disjunction
- subformula with $n$ free variables $\leadsto$ $|\textbf{adom}|^n$ gates
  $\leadsto$ especially: $|\textbf{adom}|^0 = 1$ output gate for Boolean query

# Example

We consider the formula

$$\exists z.(\exists x.\exists y.R(x,y) \wedge S(y,z)) \wedge \neg R(a,z)$$

Over the database instance:

R:

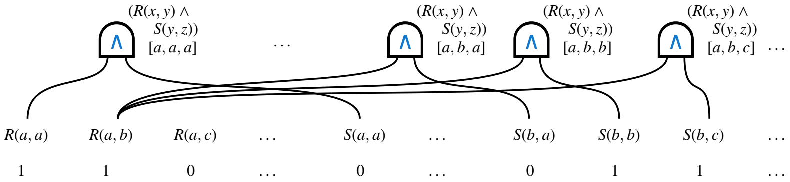
| | |
|---|---|
| $a$ | $a$ |
| $a$ | $b$ |

S:

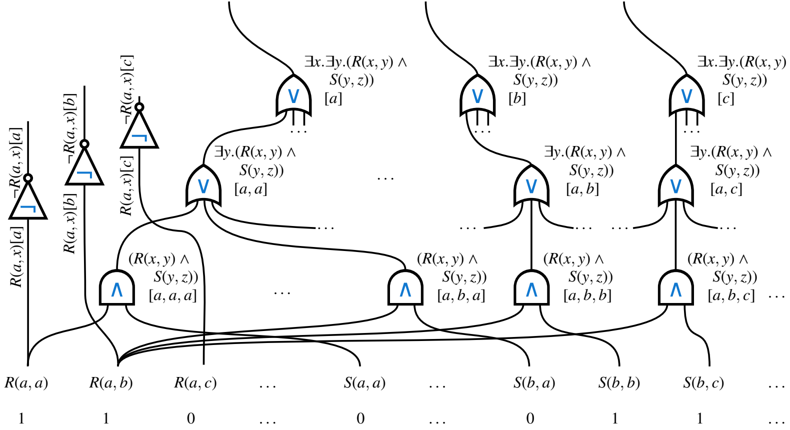
| | |
|---|---|
| $b$ | $b$ |
| $b$ | $c$ |

Active domain: $\{a, b, c\}$

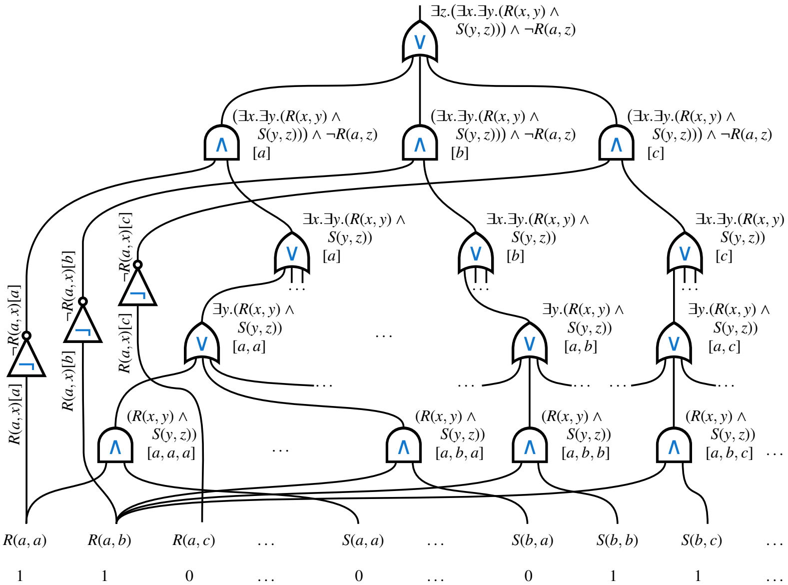
# Example: $\exists z.(\exists x.\exists y.R(x,y) \wedge S(y,z)) \wedge \neg R(a,z)$



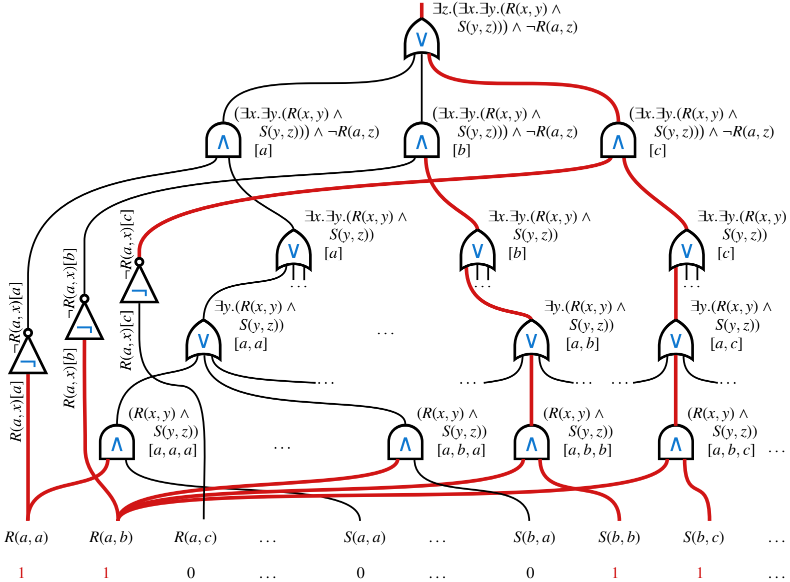
# Example: $\exists z.(\exists x.\exists y.R(x,y) \land S(y,z)) \land \neg R(a,z)$



# Example: $\exists z.(\exists x.\exists y.R(x,y) \wedge S(y,z)) \wedge \neg R(a,z)$



# Example: $\exists z.(\exists x.\exists y.R(x,y) \wedge S(y,z)) \wedge \neg R(a,z)$



# Summary and Outlook

The evaluation of FO queries is

- PSpace-complete for combined complexity
- PSpace-complete for query complexity
- $AC^0$-complete for data complexity

Circuit complexities help to identify highly parallelisable problems in P

**Open questions:**

- Are there query languages with lower complexities? (next lecture)
- Which other computing problems are interesting?
- How can we study the expressiveness of query languages?