UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Özgür L. Özçep

Finite Model Theory

Lecture 3: Games, Locality, 0-1 laws for FOL
4 November, 2015

Foundations of Ontologies and Databases
for Information Systems
CS5130 (Winter 2015)

Recap of Lecture 2: FOL

FOL as a Representation Language

- FOL provides expressive language with neat semantics to represent assertions relevant for CS
  - System descriptions
  - Desired requirements
  - System behavior description
  - Domain constraints

  - See Exercise 2.1 and Exercise 2.2

Solving Algorithmic Problems in FOL

- Definitions for important semantical properties (satisfaction, satisfiability, entailment) do not tell how to compute them

- Proof calculi to the rescue
- Various FOL calculi exist that have desired properties of being correct and complete
- Prominent ones that are "directed" and hence well implementable: Tableaux and Resolution

- Resolution calculi
  - Refutation calculus (un-satisfiability tester)
  - Data structure: Formula in Clausal Normal Form (see Exercise 2.3)
  - Resolution rule:

$$(A \vee \neg B) \wedge (C \vee B) \vDash_{res} A \vee C$$

Solving Algorithmic Problems in FOL

- No decidability for validity (unsatisfiability, entailment) but semi-decidability
- Hence we will have to consider different variants of FOL
- Undecidability stays when changing to finite model semantics
  It becomes even more worse

Theorem (Trakhtenbrot)

Validity of FOL sentences under finite model semantics is not semi-decidable

- Nonetheless FOL has important role (for CS)
  - FOL "open" (has parameters) for restrictions to more feasible fragments: number of variables, predicates, arity of predicates, complex formulae construction, quantifier nesting, quantifier alternation etc.
  - FOL (per se) is useful as a query language on DBs: constant time in data complexity ($\Longrightarrow$ to be discussed today)

End of Recap

Literature Hints

- **Lit:** L. Libkin. The finite model theory toolbox of a database theoretician. In PODS '09: Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 65–76, New York, NY, USA, 2009. ACM.
- **Lit:** L. Libkin. Elements Of Finite Model Theory (Texts in Theoretical Computer Science. An Eatcs Series). SpringerVerlag, 2004.
- **Lit:** H. Ebbinghaus and J. Flum. Finite Model Theory. Perspectives in mathematical logic. Springer, 1999.

Aim

Understand: “Finite Model Theory (FMT) is the backbone of database theory”

Finite Model Theory

- Fundamental ideas
  1. Consider DBs as finite FOL structures
  2. Consider FOL as query languages over DBs
- Starting with FOL investigate all relevant (algorithmic) problems with finite structure semantics

- These ideas are make up an approximative but nonetheless very fruitful theoretical approach to studying DB related problems
  - Showing expressivity bounds for query languages
  - Showing equivalence of DB query languages
  - Showing the inherent complexity of DB query languages

Finite Model Theory

- Fundamental ideas
  1. Consider DBs as finite FOL structures
  2. Consider FOL as query languages over DBs
- Starting with FOL investigate all relevant (algorithmic) problems with finite structure semantics
- These ideas are make up an approximative but nonetheless very fruitful theoretical approach to studying DB related problems
  - Showing expressivity bounds for query languages
  - Showing equivalence of DB query languages
  - Showing the inherent complexity of DB query languages

FOL as a Query Language

- FOL query formula $\phi(\vec{x})$ (for $\vec{x} = x_1 \ldots, x_n$) over signature σ
  - $\vec{x}$ = distinguished variables, answer variables.

Definition (Answers of a query on a structure)

$$\begin{aligned} ans(\phi(\vec{x}), \mathfrak{A}) &= \mathfrak{A}^{\phi(\vec{x})} \\ &= \{ \vec{d} = (d_1, \ldots, d_n) \mid d_i \in A \text{ and } \mathfrak{A} \models \phi(\vec{x}/\vec{d}) \} \end{aligned}$$

- Set of answer can be considered as a structure with n-ary predicate *ans*
- **n-ary query induced by ϕ:**

$$Q_\phi : STRUCT(\sigma) \longrightarrow STRUCT(ans)$$

Boolean Queries

- **Boolean FOL query formula** = FOL formulae without free variables (also called **sentences**)
- According to definition possible answers are $\{()\}$ (stands for true) and $\emptyset$ (false)
- Boolean queries can be identified with the class of σ structures making them true

Answering (Boolean) FOL queries

- Why is FOL so successful in DB theory?
- E.g., is model checking problem ($\mathfrak{A} \models \phi$) feasible?

- Answer is NO if considering $\mathfrak{A}, \phi$ both as inputs
  $\Longrightarrow$ **Combined complexity**

Theorem (Stockmeyer 74, Vardi 82)

Model-checking for FOL (and monadic second-order logic MSO) is PSPACE complete.

Lit: L. J. Stockmeyer. The complexity of decision problems in automata theory and logic. PhD thesis, MIT, 1974.

Lit: M. Y. Vardi. The complexity of relational query languages (extended abstract). In Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC '82, pages 137–146, New York, NY, USA, 1982. ACM.

Answering (Boolean) FOL queries

- Why is FOL so successful in DB theory?
- E.g., is model checking problem ($\mathfrak{A} \models \phi$) feasible?

- Answer is NO if considering $\mathfrak{A}, \phi$ both as inputs
  $\Longrightarrow$ **Combined complexity**

Theorem (Stockmeyer 74, Vardi 82)

Model-checking for FOL (and monadic second-order logic MSO) is PSPACE complete.

Lit: L. J. Stockmeyer. The complexity of decision problems in automata theory and logic. PhD thesis, MIT, 1974.

Lit: M. Y. Vardi. The complexity of relational query languages (extended abstract). In Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC '82, pages 137–146, New York, NY, USA, 1982. ACM.

Complexity Classes

- Encode algorithmic **problem** Π as a language $\Pi \subseteq \Sigma^*$, i.e., sets of words over an alphabet Σ.

- PTIME = Problems solvable in polynomial time (w.r.t. the input size) by a deterministic Turing machine
- PSPACE = Problems solvable in polynomial space (w.r.t. the input size) by a (deterministic) Turing machine

- We will come across different complexity classes
- Mostly, as computer scientist, you do not refer directly to TMs for getting complexity results
- Instead you (should) train yourself in the **art of reducing** and learning paradigmatic problems in complexity classes.

Lit: Complexity Zoo: `https://complexityzoo.uwaterloo.ca/Complexity_Zoo`

Lit: M. R. Garey and D. S. Johnson. Computers and Intractability; A Guide to the Theory of NP- Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

Complexity Classes

- Encode algorithmic **problem** Π as a language $\Pi \subseteq \Sigma^*$, i.e., sets of words over an alphabet Σ.

- PTIME = Problems solvable in polynomial time (w.r.t. the input size) by a deterministic Turing machine
- PSPACE = Problems solvable in polynomial space (w.r.t. the input size) by a (deterministic) Turing machine

- We will come across different complexity classes
- Mostly, as computer scientist, you do not refer directly to TMs for getting complexity results
- Instead you (should) train yourself in the **art of reducing** and learning paradigmatic problems in complexity classes.

Lit: Complexity Zoo: `https://complexityzoo.uwaterloo.ca/Complexity_Zoo`

Lit: M. R. Garey and D. S. Johnson. Computers and Intractability; A Guide to the Theory of NP- Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

Complete Problems

- "Paradigmatic" problems in a complexity class $\mathcal{C}$ called $\mathcal{C}$-complete problems
- $\mathcal{C}$ **complete problems** (w.r.t. $\mathcal{C}'$ reductions)=
  Most difficult problems in $\mathcal{C}$ =
  $\{\Pi \mid \Pi \in \mathcal{C}$ and all other $\mathcal{C}$ problems are $\mathcal{C}'$-reducible to $\Pi\ \}$
- Problem $\Pi \subseteq \Sigma$ is $\mathcal{C}$-**reducible** to problem $\Pi' \subseteq \Sigma'$, for short: $\Pi \leq_{\mathcal{C}} \Pi'$, iff there is a $\mathcal{C}$-computable function such that for all $w \in \Sigma$: $w \in \Pi$ iff $f(w) \in \Pi'$

Example for PSPACE Complete Problem

- Quantified Boolean Formula (QBF)
  - All propositional symbols p_i are QBF
  - All boolean combinations of QBFs are QBFs
  - If ϕ is a QBF, then so are$\forall p\phi$ and $\exists p\phi$.
  - Semantics: Structures here are truth value assignments

- Theorem: Satisfiability of QBFs is PSPACE complete

Example

- $\exists p \exists q\ p \wedge q$ is satisfiable, because
  there is assignment $\nu(q) = 1$ and $\nu(q) = 1$ making $p \wedge q$ true.
- $\exists p\ p \wedge \neg p$ is not satisfiable

FOL is in PSPACE

Complexity estimation for query answering

Time complexity for checking $\mathfrak{A} \models \phi$ is $O(n^k)$, where

- n = size of input structure $\mathfrak{A}$ and
- k = size of input query ϕ

Note: Size of query k is responsible for exponential blow up

Reminder (Landau-Notation)

- $f \in O(g)$ means: f has function g as **upper bound**
- Formally: There are constants $c > 0$ and x_0, s.t. for all $x > x_0$:

$$|f(x)| \leq c * |g(x)|$$

FOL is in PSPACE

Complexity estimation for query answering

Time complexity for checking $\mathfrak{A} \models \phi$ is $O(n^k)$, where

- n = size of input structure $\mathfrak{A}$ and
- k = size of input query ϕ

- **Note**: Size of query k is responsible for exponential blow up

- Naive recursive algorithm showing time complexity $(O(n^k))$ and space complexity $(O(k * log(n)))$
  - Atomic formula: Look up in structure
  - Boolean cases: apply semantics of Boolean connectors
  - $\exists x \phi(x)$: Check for all $d \in A$ whether $\mathfrak{A} \models \phi(x/d)$

- PSPACE completeness by reducing QBF satisfiability to FOL model checking

FOL is in AC^0

- In practical scenarios DB size n much bigger than query size k
- Therefore: Consider only DB as input; query fixed
  $\Longrightarrow$ **data complexity**

- This helps al lot, as only query size responsible for exponential complexity, indeed:

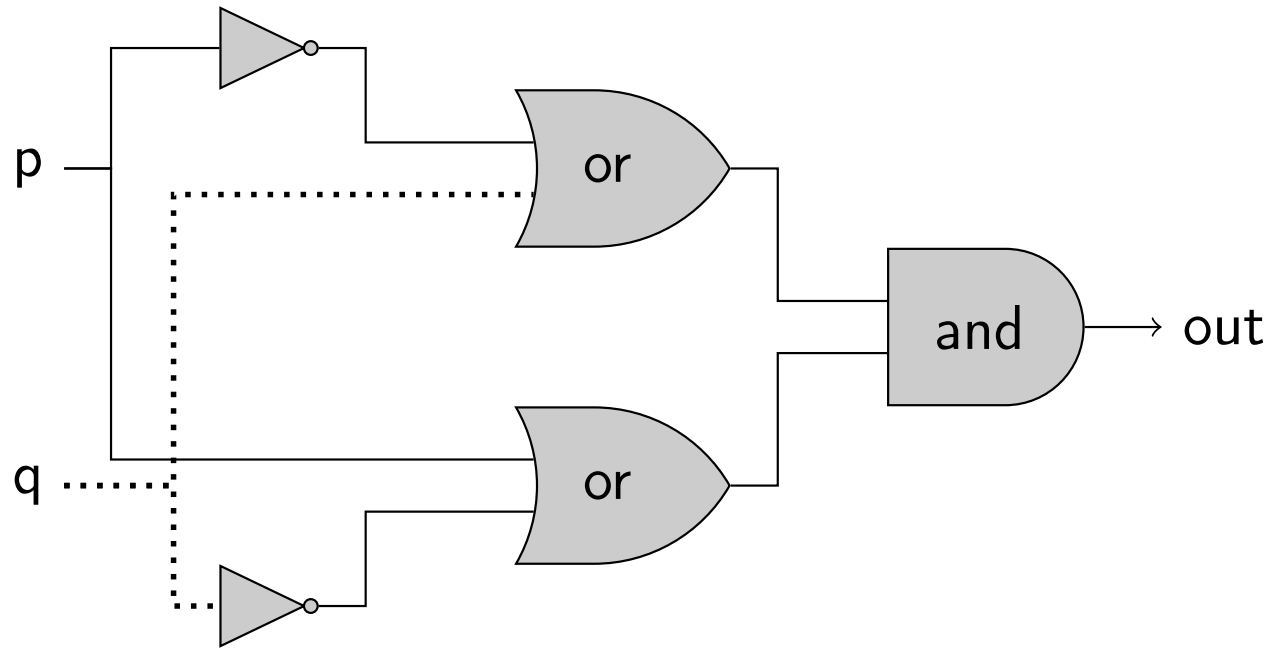
Theorem

FOL is in LOGSPACE and even in AC^0.

- LOGSPACE = Problems solvable in logarithmic space on the read-write tape by a deterministic 2-tape Turing machine
- $AC^0 \subsetneq LOGSPACE$.

The Class AC^0

- Intuitively AC^0 = class of problems solvable in constant time on polynomially many process (in parallel)
- Formally AC^0 defined using computation model based on boolean circuits



Boolean circuit above computes $(\neg p \vee q) \wedge (p \wedge \neg q)$

The class AC^0

- Encode problems as 0/1 vector inputs
- Computability by circuits: There is family of circuits (for every possible size of input) computing desired boolean function

- In many cases: uniformity condition: family not arbitrarily constructed by computable

Definition

AC^0 = Problems solvable by families of circuits with

- constant depth,
- polynomial size and
- using NOT gates, unlimited-fanin AND gates and OR gates.

FOL is in AC^0 data complexity

Proof idea

- Query modelled as boolean circuit family for every possible instance of given DB schema $\mathcal{R}$ and super-domain Dom
- Every ground atom $R(d_1, \ldots, d_n)$ is represented as propositional input symbol
- Gates for every subexpression of query
- Boolean operators in subexpression modelled by corresponding boolean gates
- $\exists$ ($\forall$) quantifier modelled by unbounded fan-in OR (AND) gate

Lit: S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. 1995.

Proving Expressivity Bounds for FOL

Expressivity of Languages

- We defined above the query Q_ϕ induced by a formula ϕ (syntax-> semantics direction)

- For expressivity considerations one goes the other way round (semantics -> syntax):
  - Given a query

$$Q : STRUC(\sigma) \longrightarrow STRUC(ans)$$

    test whether there is formula ϕ in the given logic s.t. $Q = Q_\phi$
  - In this case one says that Q is **definable** in the logic (for the given set of structures $STRUC(\sigma)$)

- For Boolean queries definability amounts to:
  Given a class $X \subseteq STRUC(\sigma)$ of structures over a signature σ:
  there is a sentence ϕ (over the given logic) s.t. $Mod(\phi) = X$

Expressivity of Languages

- We defined above the query Q_ϕ induced by a formula ϕ (syntax-> semantics direction)

- For expressivity considerations one goes the other way round (semantics -> syntax):
  - Given a query

$$Q : STRUC(\sigma) \longrightarrow STRUC(ans)$$

    test whether there is formula ϕ in the given logic s.t. $Q = Q_\phi$
  - In this case one says that Q is **definable** in the logic (for the given set of structures $STRUC(\sigma)$)

- For Boolean queries definability amounts to:
  Given a class $X \subseteq STRUC(\sigma)$ of structures over a signature σ:
  there is a sentence ϕ (over the given logic) s.t. $Mod(\phi) = X$

Expressivity of Languages

- We defined above the query Q_ϕ induced by a formula ϕ (syntax-> semantics direction)

- For expressivity considerations one goes the other way round (semantics -> syntax):
  - Given a query

    $$Q : STRUC(\sigma) \longrightarrow STRUC(ans)$$

    test whether there is formula ϕ in the given logic s.t. $Q = Q_\phi$
  - In this case one says that Q is **definable** in the logic (for the given set of structures $STRUC(\sigma)$)

- For Boolean queries definability amounts to:
  Given a class $X \subseteq STRUC(\sigma)$ of structures over a signature σ:
  there is a sentence ϕ (over the given logic) s.t. $Mod(\phi) = X$

Need for New Proof Techniques

- Main classical techniques used for classical FOL do not work
- Because corresponding theorems do not hold for FMT

- Reminder: Main properties of FOL
  - Compactness (Comp)
  - Löwenheim-Skolem (Lösko)
- These properties characterize FOL for arbitrary structures: Lindström theorems

Finite Compactness Pendant?

Fin-Comp

If every finite subset of Φ has a finite model, then Φ has a finite model.

- The finite version of compactness (Fin-Comp) does not hold for FOL.
- Falsifier
  - $\lambda_n := \exists x_1, \ldots, x_n \bigwedge_{i \neq j} \neg(x_i = x_j)$
    (says: "There are at least n elements")
  - $\{\lambda_n \mid n \in \mathbb{N}\}$ has not finite model though every subset has
    (Compare Exercise 2.2.3)

What's the right "proof technique"?

Cartoon of Sidney Harris: you want proof

We Prefer to Proof/Argue ... without Being a Poser ...

Pinguin Video

URL: `https://www.youtube.com/watch?v=7iDn5d9q9Y8`

Convention for the Following

Assume all structures are relational, i.e., there are no function symbols other than constants—unless stated otherwise

Games

Games as Essence of Being a Human

"Der Mensch spielt nur, wo er in voller Bedeutung des Wortes Mensch ist, und er ist nur da ganz Mensch, wo er spielt"

(F. Schiller, Briefe über die ästhetische Erziehung des Menschen (1795))

Games as a CS Tool

- In logic Fraïssé games are an important proof tool
- Different variations (w.r.t. rules, winning strategies)
- We will consider a basic game type and show how to use it.

- But: games have high "cognitive complexity" even for non-trivial problems
- Therefore: Use games for simple but generic problems and **reduce** others to these

Ehrenfeucht-Fraïssé Games

- $G_n(\mathfrak{A}, \mathfrak{B})$
  n-round game played for structures on same signature

- Input: structures $\mathfrak{A}, \mathfrak{B}$

- Two players: spoiler and duplicator
- Finite number of rounds n

- Output: a function relating elements from $\mathfrak{A}, \mathfrak{B}$

- Spoiler's aim: show $\mathfrak{A}, \mathfrak{B}$ are "different"
- Duplicator's aim: show $\mathfrak{A}, \mathfrak{B}$ are "same"

Rules of the Game

- In turn, spoiler choose structure and element i in it and
- duplicator chooses other structure and element in it

- After n rounds: n elements $a_1, \ldots, a_n$ from $\mathfrak{A}$ and n elements $b_1, \ldots, b_n$ from $\mathfrak{B}$ are chosen.

Winning condition
Duplicator wins iff
$(a_1, \ldots, a_n)$ plays in $\mathfrak{A}$ the same role as $(b_1, \ldots, b_n)$ in $\mathfrak{B}$

Partial Isomorphism

Formalize sameness of tuples' roles by notion of partial isomorphism

Definition (Partial Isomorphism)

For structures $\mathfrak{A}$, $\mathfrak{B}$ over signature σ, let $f : A \longrightarrow B$ a function with domain $dom(f)$. f is called a **partial isomorphism** iff

- f is injective
- For every constant c: $c^{\mathfrak{A}} \in dom(p)$ and $f(c^{\mathfrak{A}}) = c^{\mathfrak{B}}$
- For all relation symbols R (including identity) and $a_1, \ldots, a_n \in dom(f)$
  $R^{\mathfrak{A}}(a_1, \ldots, a_n)$ iff $R^{\mathfrak{B}}(f(a_1), \ldots, f(a_n))$

If f is total and bijective, then f is called an **isomorphism**, and $\mathfrak{A}, \mathfrak{B}$ are said to be isomorphic, for short $\mathfrak{A} \simeq \mathfrak{B}$

Winning Condition Formalized

- After n rounds: n elements $a_1, \ldots, a_n$ from $\mathfrak{A}$ and n elements $b_1, \ldots, b_n$ from $\mathfrak{B}$ are chosen.

- **Winning condition**
  Duplicator wins iff
  $f : a_i \mapsto b_i$ is a partial isomorphism of $\mathfrak{A}$ and $\mathfrak{B}$.

- **Game equivalence**
  $\mathfrak{A} \sim_{G_n} \mathfrak{B}$ iff: Duplicator has a winning strategy in $G_n(\mathfrak{A}, \mathfrak{B})$
  ($\mathfrak{A}$ and $\mathfrak{B}$ are the same w.r.t. n-round games)

Quantifier Rank

- How do we use games for proving in-expressivity?
- We need two more technical notions
  - to capture nesting depth of quantifiers
  - to capture property that two structures model the same sentences (up to some syntactical complexity)

Quantifier Rank

- How do we use games for proving in-expressivity?

- We need two more technical notions
  - to capture nesting depth of quantifiers
  - to capture property that two structures model the same sentences (up to some syntactical complexity)

Definition (Quantifier Rank)

The quantifier rank of a formulae ϕ, for short $qr(\phi)$, is defined recursively as follows:

- $qr(\phi) = 0$ for atoms ϕ
- $qr(\phi \vee \psi) = qr(\phi \wedge \psi) = qr(\phi \rightarrow \psi) = max\{qr(\phi), qr(\psi)\}$
- $qr(\neg\phi) = qr(\phi)$
- $qr(\exists x\ \phi) = qr(\forall x\ \phi) = qr(\phi) + 1$

Quantifier Rank

- How do we use games for proving in-expressivity?
- We need two more technical notions
  - to capture nesting depth of quantifiers
  - to capture property that two structures model the same sentences (up to some syntactical complexity)

Definition (Equivalence Up to Rank n)

$\mathfrak{A} \equiv_n \mathfrak{B}$ iff $\mathfrak{A}$ and $\mathfrak{B}$ agree on all FOL sentences of quantifier rank up to n.

How to Use Games?

Theorem

$\mathfrak{A} \sim_{G_n} \mathfrak{B}$ *iff* $\mathfrak{A} \equiv_n \mathfrak{B}$

This gives a **non-FOL-expressibility tool**

- Aim: Show Q not expressible in FOL
- Construct families of structure $\mathfrak{A}_n, \mathfrak{B}_n$ s.t.
  1. All $\mathfrak{A}_n$ satisfy Q
  2. No $\mathfrak{B}_n$ satisfies Q
  3. $\mathfrak{A}_n \sim_{G_n} \mathfrak{B}_n$
- Assume Q expressible as FOL formula ϕ of quantifier rank n.

  Then $\mathfrak{A}_n \models \phi$ and $\mathfrak{B}_n \models \neg\phi$, but $\mathfrak{A}_n \sim_{G_n} \mathfrak{B}_n$

Example: Inexpressibility of EVEN

- **EVEN(σ)**: structures over signature σ with domain of even cardinality
- The signature is relevant for the proofs
- Simpel case: $\sigma = \emptyset \quad \Longrightarrow \quad$ structures are sets

Proposition

EVEN($\emptyset$) is not expressible in FOL

Proof

- Choose $\mathfrak{A}_n$ as $2n$-element set, $\mathfrak{B}_n$ as $2n+1$-element set.
- $\mathfrak{A}_n \in EVEN(\emptyset)$ and $\mathfrak{B}_n \notin EVEN(\emptyset)$
- $\mathfrak{A}_n \sim_{G_n} \mathfrak{B}_n$: Duplicator plays already played element in the other set iff spoiler does

Example: Inexpressibility of EVEN

- **EVEN(σ)**: structures over signature σ with domain of even cardinality
- The signature is relevant for the proofs
- Simpel case: $\sigma = \emptyset \quad \Longrightarrow \quad$ structures are sets

Proposition

EVEN($\emptyset$) is not expressible in FOL

Proof

- Choose $\mathfrak{A}_n$ as $2n$-element set, $\mathfrak{B}_n$ as $2n+1$-element set.
- $\mathfrak{A}_n \in \mathit{EVEN}(\emptyset)$ and $\mathfrak{B}_n \notin \mathit{EVEN}(\emptyset)$
- $\mathfrak{A}_n \sim_{G_n} \mathfrak{B}_n$: Duplicator plays already played element in the other set iff spoiler does

Inexpressibility of EVEN(σ) with Games

- What about EVEN(σ) for non-empty σ?
- Consider: $\sigma = \{<\}$ and class of structures = linear orders
- L_n: n-element total ordering on some set

Theorem

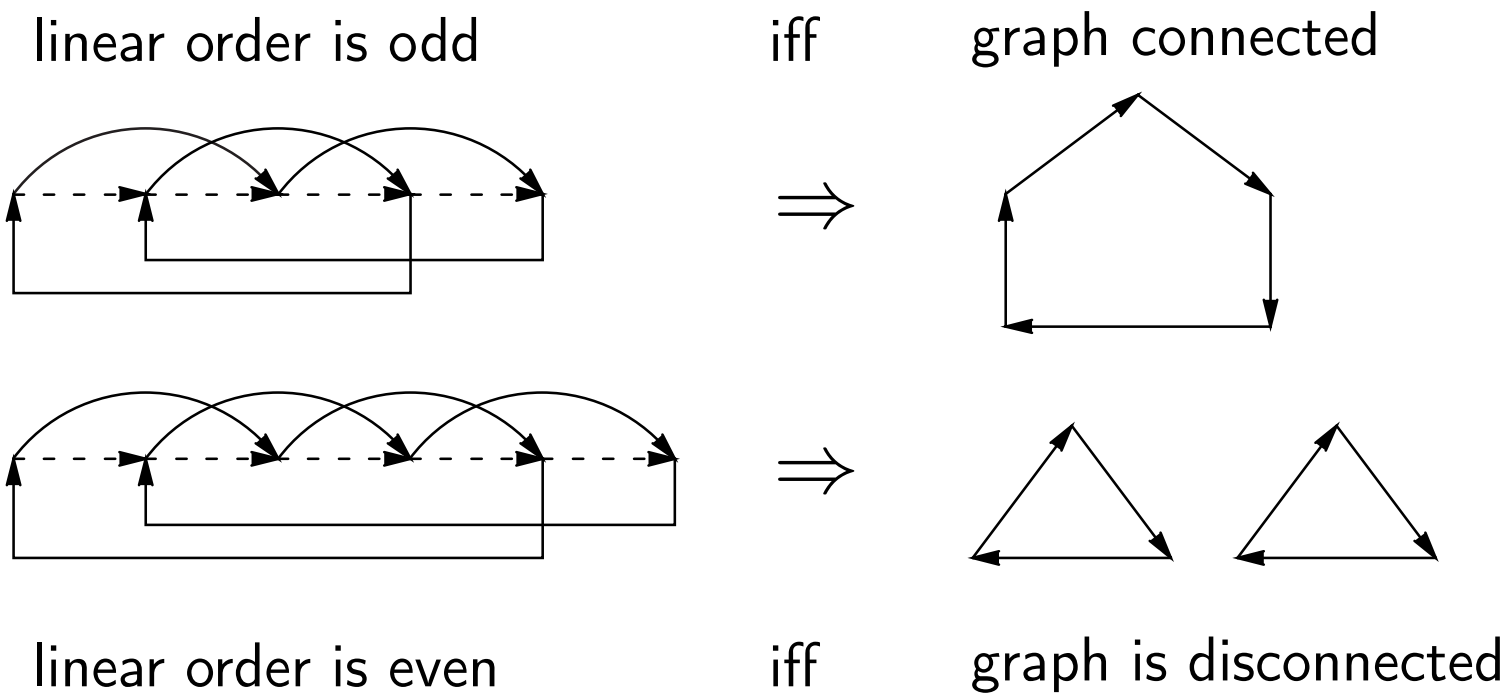
For every $m, k \geq 2^n$: $L_m \sim_{G_n} L_k$.

- In particular *EVEN*($<$) not expressible over linear orders: take $\mathfrak{A}_n = L_{2^n}$, $\mathfrak{B}_n = L_{2^{n+1}}$.

Proving Inexpressivity: Reduction Tricks (not Tools)

- Showing FOL inexpressibility of
  - graph connectivity CONN
  - acyclicity ACYCL
  - transitive closure TC
- by reduction of EVEN($<$)

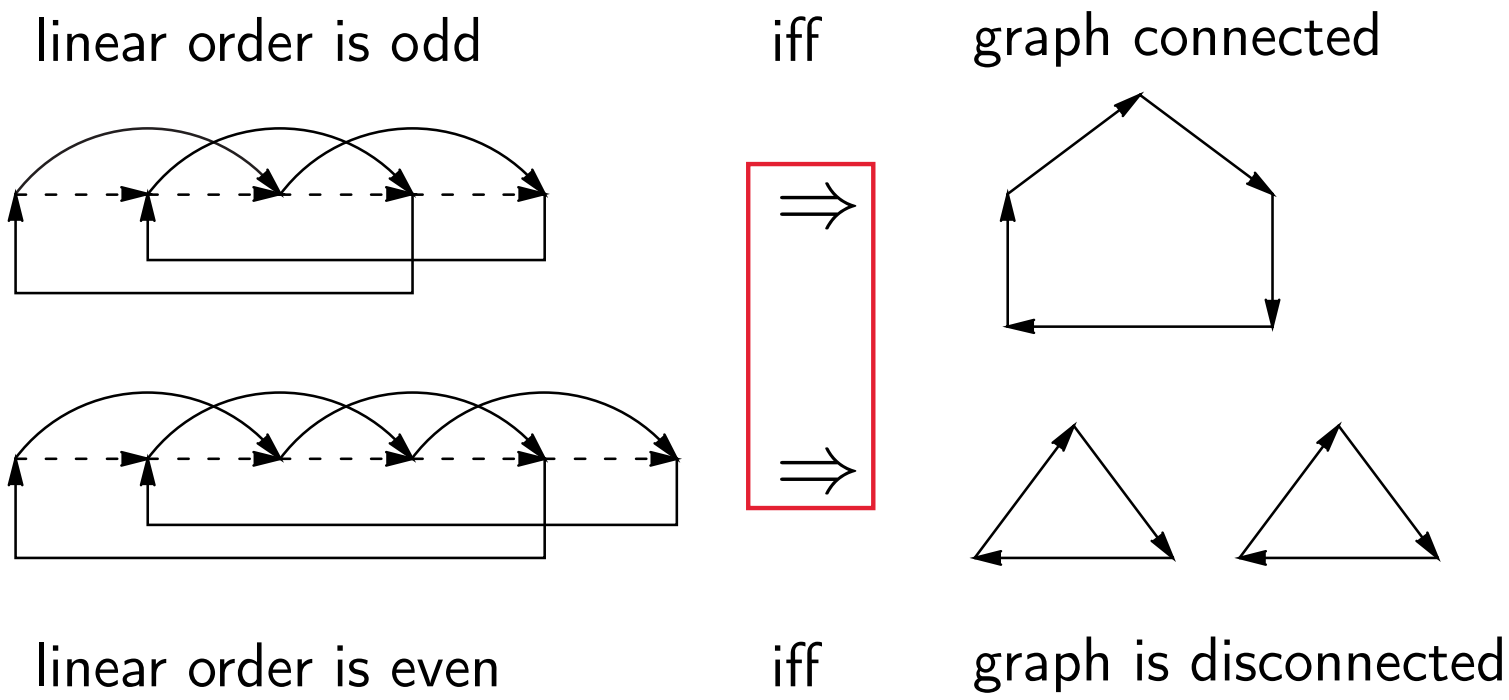
Reduce EVEN(<) to Graph Connectivity



- Construction of graph from linear order expressible as an FOL query $Q_{red} : LinOrd \longrightarrow GRAPH$

Graphics from [Libkin 09]

Reduce EVEN(<) to Graph Connectivity



- Construction of graph from linear order expressible as an FOL query $Q_{red} : LinOrd \longrightarrow GRAPH$

Graphics from [Libkin 09]

ACYCL and TC are not FOL expressible

- ACYCL: Reduction EVEN $\Rightarrow$ ACYCL as above but with one back edge from last node to first node
- Reduction for TC: CONN $\Rightarrow$ TC
  - Add edge $E(x, y)$ for every edge $E(y, x)$
  - Compute TC on resulting graph
  - Test whether graph is complete

Locality

Proving Inexpressibility by Locality

- FOL has a fundamental property: locality
- Consider a binary query $Q : STRUCT(\sigma) \longrightarrow STRUC(ans)$
  - Need a formula ϕ_Q in two open variables x, y
  - The way how to describe constraints between x and y is restricted by the number of atoms and elements occurring in ϕ_Q.
- Different (comparable) locality notions
  - Bounded number of degrees property (BNDP)
  - Gaifman locality
  - Hanf locality

Proving Inexpressibility by Locality

- FOL has a fundamental property: locality
- Consider a binary query $Q : STRUCT(\sigma) \longrightarrow STRUC(ans)$
  - Need a formula ϕ_Q in two open variables x, y
  - The way how to describe constraints between x and y is restricted by the number of atoms and elements occurring in ϕ_Q.
- Different (comparable) locality notions
  - Bounded number of degrees property (BNDP)
  - Gaifman locality
  - Hanf locality

BNDP

- $in(\mathfrak{G})$ = set of in-degrees of nodes in $\mathfrak{G}$
- $out(\mathfrak{G})$ = set of out-degrees of nodes in $\mathfrak{G}$
- $degs(\mathfrak{G}) = in(\mathfrak{G}) \cup out(\mathfrak{G})$

Definition

Q has BNDP iff there is $f_Q : \mathbb{N} \longrightarrow \mathbb{N}$ s.t. for all graphs $\mathfrak{G}$:

If there is $k \in \mathbb{N}$ s.t. $\max(degs(\mathfrak{G})) \leq k$,
then $|degs(Q(\mathfrak{G}))| \leq f_Q(k)$.

- Intuitively: Q disallowed to arbitrarily increase degrees of nodes

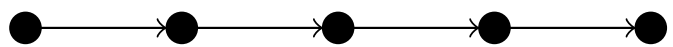
Theorem

Every FOL query has the BNDP.

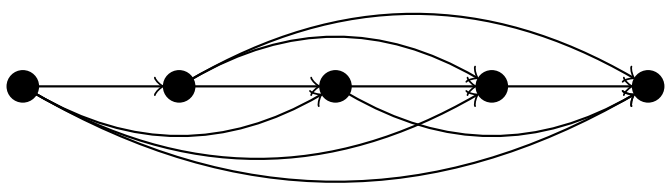
Example: TC on Successor Relation Graph

- $\mathfrak{G} = (\{a_0, \ldots, a_n\}, \{E(a_0, a_1), \ldots, E(a_{n-1}, a_n)\})$
- $in(\mathfrak{G}) = out(\mathfrak{G}) = \{0, 1\}$
- $in(TC(\mathfrak{G})) = out(TC(\mathfrak{G})) = \{0, \ldots, n-1\}$

$\mathfrak{G}$



$TC(\mathfrak{G})$



Gaifman locality

Gaifman locality defined here on graphs $\mathfrak{G} = (A, E)$
(can be generalized to arbitrary structures)

Gaifman Locality Intuitively

An m-ary query Q is Gaifman local iff there is a threshold (radius) r such that for all graphs:

Q cannot distinguish between tuples if their r-neighbourhoods in the graph are the same.

Theorem

Every FOL-definable query is Gaifman-local.

Gaifman locality

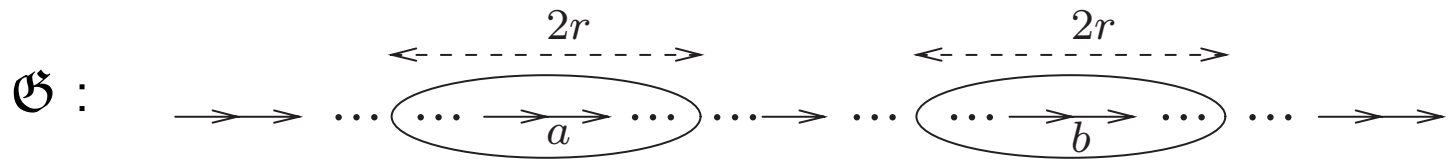
- $\overline{a} \in A^n$ (vector of elements)
- $B_r^{\mathfrak{G}}(\overline{a}) = \{b \in A \mid d(\overline{a}, b) \leq r\}$ **(radius r ball around $\overline{a}$)**
  $d(\overline{a}, b))$ = minimal path distance from $\{a_1, \ldots a_n\}$ to b
- $N_r^{\mathfrak{G}}(\overline{a})$ **(r-neighbourhood of $\overline{a}$)**
  subgraph induced by $B_r^{\mathfrak{G}}(\overline{a})$ in the structure $(A, E, \overline{a})$

Definition

An m-ary query Q (with $m > 0$) is **Gaifman-local** iff:

There exists a radius r s.t. for all $\mathfrak{G}$: If $N_r^{\mathfrak{G}}(\overline{a}) \simeq N_r^{\mathfrak{G}}(\overline{b})$, then $\overline{a} \in Q(\mathfrak{G})$ exactly when $\overline{b} \in Q(\mathfrak{G})$.

Example: TC is not Gaifman local



Proof

- Suppose TC is FOL definable with query Q
- Then Q is Gaifman local with some radius r
- $N_r^{\mathfrak{G}}((a,b)) \simeq N_r^{\mathfrak{G}}((b,a))$
  because both subgaphs are disjoint unions of two 2r-chains
- But $(a,b) \in TC(\mathfrak{G})$ and $(b,a) \notin TC(\mathfrak{G})$,

Graphics from [Libkin 09]

Hanf locality

- $\mathfrak{G} = (A, E), \mathfrak{G}' = (A', E')$
- $\mathfrak{G} \rightleftarrows_r \mathfrak{G}'$ iff
  there exists bijection $f : A \longrightarrow A'$ s.t. for all $a \in A$:
  $N_r^{\mathfrak{G}}(a) \simeq N_r^{\mathfrak{G}'}(f(a))$
- Intuitively: $\mathfrak{G}, \mathfrak{G}'$ are pointwise similar w.r.t. to r-neighbourhoods

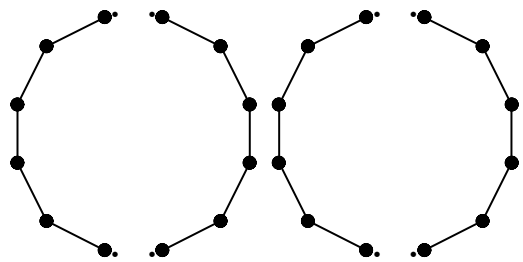
Definition

A Boolean query Q is **Hanf-local** iff a radius r exists s.t. for any graphs $\mathfrak{G}, \mathfrak{G}'$ with $\mathfrak{G} \rightleftarrows_r \mathfrak{G}'$ one has $Q(\mathfrak{G}) = Q(sG')$.

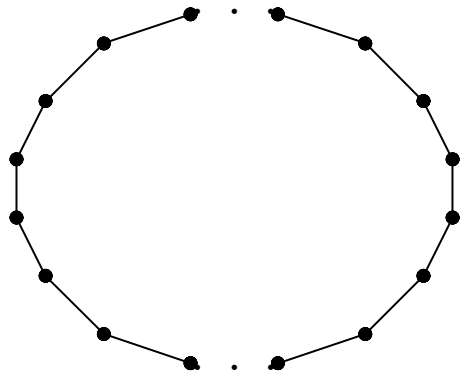
Theorem

Every FOL definable Boolean query is Hanf-local.

Example: CONN is not Hanf-local



G: two cycles of length m



G': one cycle of length $2m$

Proof

- For contradiction assume CONN is Hanf-local with parameter r
- Choose $m > 2r + 1$; f an arbitrary bijection of G and G'
- r-neighbourhood of any a the same: 2r-chain with a in the middle
- Hence $G \rightleftarrows_r G'$, but: G' is connected and G is not.

Graphics from [Libkin 09]

Comparison of Locality Notions

Theorem

Hanf local $\models$ *Gaifmann local* $\models$ *BNDP*

0-1 law

0-1 law

An inexpressibility tool based on a probabilistic property of FOL queries

0-1-law informally

Either almost all finite structures fulfill the property or almost all do not

Example

- Boolean query $Q_1 = \forall x, y\ E(x, y)$ on graphs
  Almost all graphs do not satisfy Q_1 (only the complete ones)

- Boolean query $Q_2 = \forall x \forall y \exists z\ E(z, x) \wedge \neg E(z, y)$
  Almost all graphs satisfy Q_2

Formal definition 0-1 laws

- Here it is important that signature σ is relational!!
- $STRUC(\sigma, n)$: structures with domain $[n] := \{0, 1, \ldots, n-1\}$ over σ.
- For a Boolean query Q let

$$\mu_n = \frac{|\{\mathfrak{A} \in STRUC(\sigma, n) \mid Q(\mathfrak{A}) = true\}|}{|STRUC(\sigma, n)|}$$

- $\mu_n(Q)$ is the probability that a randomly chosen structure on $[n]$ satisfies Q
- $\mu(Q) = \lim_{n\to\infty} \mu_n(Q)$

Definition

A logic has 0-1-law if for every Boolean query Q expressible in it either $\mu(Q) = 0$ or $\mu(Q) = 1$.

Inexpressibility with 0-1 laws

Theorem

FOL has the 0-1- law.

- Helpful for proving inexpressibility of counting properties

Example (EVEN is not expressible in FOL)

$\mu(EVEN)$ not defined because $\mu_n(EVEN)$ alternates between 0 and 1.

Exercise 3 (15 Points)

Hand in your exercise as one pdf File in Moodle by November 10, 23:55h

Exercise 3.1 (4 Points)

Give at least two aspects of real DBs for which the approach of identifying DBs with finite FOL structures is not sufficient or adequate.

Exercise 3.2 (4 Points)

Argue why the usual restriction in FMT to consider only relational structures (i.e., no functions symbols allowed) is not problematic. That is, how can formulas with function symbols be represented by formulas containing only relation symbols (in particular the identity relation.)

Exercise 3.3 (4 Points)

Formalize the reduction query $Q_{red} : \mathit{LinOrd} \to \mathit{GRAPH}$ from linear orders to graphs by describing the a query formula inducing Q_{red}.

Bonus: Exercise 3.4 (3 Points)

Show using the 0-1 law that the property $D_k =$ "structures whose domain is divisible by k", for $k \in \mathbb{N} \setminus 0$, is not FOL expressible.