TECHNISCHE UNIVERSITÄT DRESDEN

International Center for Computational Logic

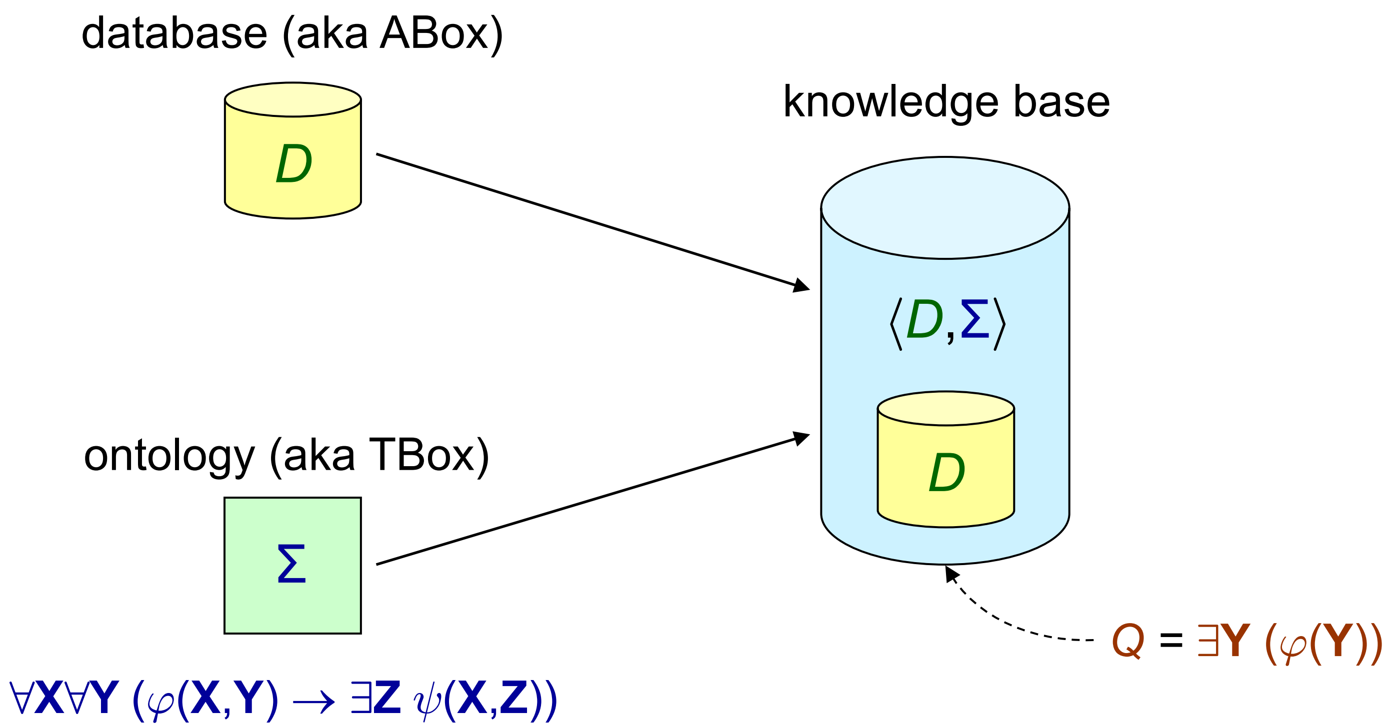
Computational Logic ∴ Group

Sebastian Rudolph
International Center for Computational Logic
TU Dresden

Existential Rules – Lecture 6

Adapted from slides by Andreas Pieris and Michaël Thomazo
Summer Term 2023

BCQ-Answering: Our Main Decision Problem



decide whether $D \wedge \Sigma \models Q$

Query Answering via the Chase

Theorem: $D \wedge \Sigma \models Q$ iff $U \models Q$, where U is a universal model of $D \wedge \Sigma$

+

Theorem: chase(D, Σ) is a universal model of $D \wedge \Sigma$

=

Corollary: $D \wedge \Sigma \models Q$ iff chase(D,Σ) $\models Q$

Undecidability of BCQ-Answering

Theorem: BCQ-Answering is undecidable

Proof : By simulating a deterministic Turing machine with an empty tape

…syntactic restrictions are needed!!!

Termination of the Chase

- Drop the existential quantification
  - We obtain the class of full existential rules
  - Very close to Datalog

- Drop the recursive definitions
  - We obtain the class of acyclic existential rules
  - A.k.a. non-recursive existential rules

Termination of the Chase

- Drop the existential quantification
  - We obtain the class of full existential rules
  - Very close to Datalog

✓

- Drop the recursive definitions
  - We obtain the class of acyclic existential rules
  - A.k.a. non-recursive existential rules

Acyclic Existential Rules

- The definition of a predicate *P* does not depend on *P* - formal definition via the predicate graph

- The predicate graph of a set Σ of existential rules, denoted PG(Σ), is the graph (V,E), where
  - $V = \{P \mid P \in \mathrm{sch}(\Sigma)\}$
  - $E = \{(P,R) \mid \forall X \forall Y\, (\ldots \wedge P(X,Y) \wedge \ldots \rightarrow \exists Z\, (\ldots \wedge R(X,Z) \wedge \ldots)) \in \Sigma\}$

$$\forall X\, (\mathit{Person}(X) \rightarrow \exists Y\, (\mathit{hasParent}(X,Y) \wedge \mathit{Person}(Y)))$$

Acyclic Existential Rules

- The definition of a predicate *P* does not depend on *P* - formal definition via the predicate graph

- The predicate graph of a set Σ of existential rules, denoted PG(Σ), is the graph (V,E), where
  - $V = \{P \mid P \in sch(\Sigma)\}$
  - $E = \{(P,R) \mid \forall X \forall Y (\ldots \wedge P(X,Y) \wedge \ldots \rightarrow \exists Z (\ldots \wedge R(X,Z) \wedge \ldots)) \in \Sigma\}$

- A set Σ of existential rules is acyclic if the graph PG(Σ) is acyclic

- We denote ACYCLIC the class of acyclic existential rules

The Naïve Algorithm for **ACYCLIC**

- The naïve algorithm shows that BCQ-Answering under **ACYCLIC** is
  - in PTIME w.r.t. the data complexity
  - in 2EXPTIME w.r.t. the combined complexity

…can we do better than the naïve algorithm?

YES!!!

Combined Complexity of ACYCLIC

Theorem: BCQ-Answering under ACYCLIC is in NEXPTIME w.r.t. the combined complexity

Proof: We first need to establish the so-called small witness property

Combined Complexity of ACYCLIC

Theorem: BCQ-Answering under ACYCLIC is in NEXPTIME w.r.t. the combined complexity

Proof: Guess-and-check, using the so-called small witness property

We cannot do better than the previous algorithm:

Theorem: BCQ-Answering under ACYCLIC is NEXPTIME-hard w.r.t. the combined complexity

Proof : By reduction from a tiling problem, a classical NEXPTIME-hard problem

Tiling Problem

Tiling:

Input: T = {$t_0,\ldots,t_k$}, a set of square tile types,

H,V $\subseteq$ T $\times$ T, the horizontal and vertical compatibility relations

n, an integer in unary

Question: decide whether a $2^n \times 2^n$ tiling exists, that is,

	1	2	3	$\cdots$	2^n
1					
2					
3					
$\vdots$					
2^n					

Tiling Problem

Tiling:

Input: $T = \{t_0,\ldots,t_k\}$, a set of square tile types,

$H,V \subseteq T \times T$, the horizontal and vertical compatibility relations

n, an integer in unary

Question: decide whether a $2^n \times 2^n$ tiling exists, that is,

$(1,1) = t_0$

	1	2	3	$\cdots$	2^n
1	t_0				
2					
3					
$\vdots$					
2^n					

Tiling Problem

Tiling:

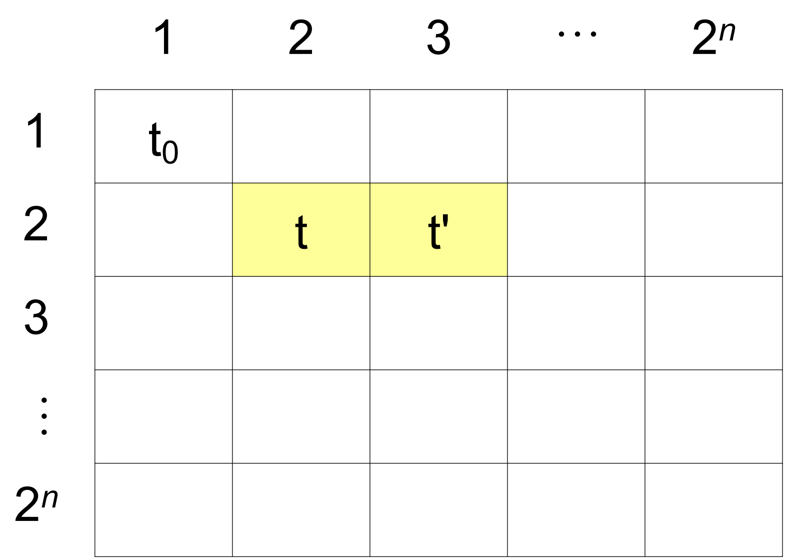
Input: $T = \{t_0,\ldots,t_k\}$, a set of square tile types,

$H,V \subseteq T \times T$, the horizontal and vertical compatibility relations

n, an integer in unary

Question: decide whether a $2^n \times 2^n$ tiling exists, that is,

$(1,1) = t_0$



$(t,t') \in H$

Tiling Problem

Tiling:

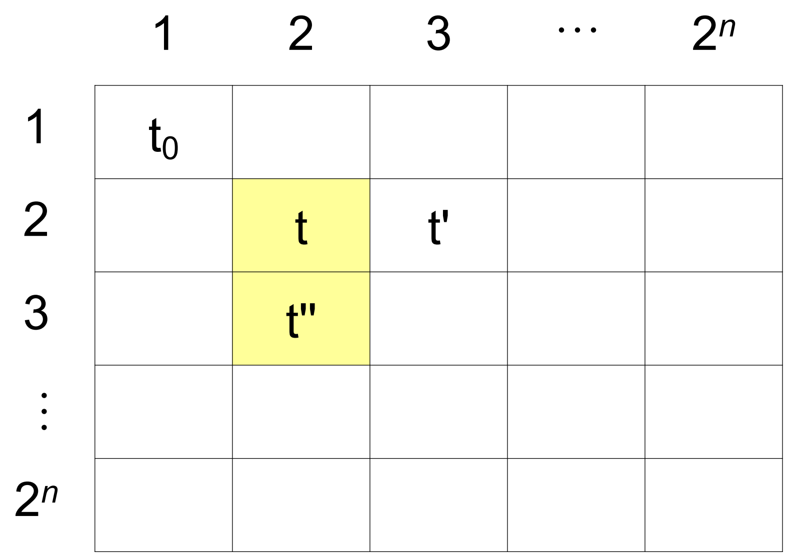
Input: $T = \{t_0,\ldots,t_k\}$, a set of square tile types,

$H,V \subseteq T \times T$, the horizontal and vertical compatibility relations

n, an integer in unary

Question: decide whether a $2^n \times 2^n$ tiling exists, that is,

$(1,1) = t_0$



$(t,t') \in H$

$(t,t'') \in V$

Combined Complexity of ACYCLIC

We cannot do better than the previous algorithm

Theorem: BCQ-Answering under ACYCLIC is NEXPTIME-hard w.r.t. the combined complexity

Proof : By reduction from a tiling problem, a classical NEXPTIME-hard problem

NEXPTIME-hardness of ACYCLIC

- The database stores the horizontal and the vertical relations

$$D = \{H(t,t') \mid (t,t') \in H\} \cup \{V(t,t') \mid (t,t') \in V\}$$

- We use $\Sigma \in$ ACYCLIC to inductively construct $2^k \times 2^k$ tilings from $2^{k-1} \times 2^{k-1}$ tilings
- The key observation is that

X_1	X_2	Y_1	Y_2
X_3	X_4	Y_3	Y_4
Z_1	Z_2	W_1	W_2
Z_3	Z_4	W_3	W_4

is a $2^k \times 2^k$ tiling

iff

X_1	X_2	X_2	Y_1	Y_1	Y_2
X_3	X_4	X_4	Y_3	Y_3	Y_4
X_3	X_4	X_4	Y_3	Y_3	Y_4
Z_1	Z_2	Z_2	W_1	W_1	W_2
Z_1	Z_2	Z_2	W_1	W_1	W_2
Z_3	Z_4	Z_4	W_3	W_3	W_4

are $2^{k-1} \times 2^{k-1}$ tilings

NEXPTIME-hardness of ACYCLIC

The $2^k \times 2^k$ tiling

X_1	X_2
X_3	X_4

is represented by an atom of the form

ID of the tiling

$T_k(S, O, X_1, X_2, X_3, X_4)$

origin of the tiling, i.e., the upper-left tile

NEXPTIME-hardness of ACYCLIC

Base step - construct 2×2 tilings of the form

X_1	X_2
X_3	X_4

$$\forall X_1 \forall X_2 \forall X_3 \forall X_4\ (H(X_1,X_2) \wedge H(X_3,X_4) \wedge V(X_1,X_3) \wedge V(X_2,X_4) \rightarrow \exists Y\ T_1(Y,X_1,X_1,X_2,X_3,X_4))$$

NEXPTIME-hardness of ACYCLIC

Inductive step - construct $2^k \times 2^k$ tilings from $2^{k-1} \times 2^{k-1}$ tilings

X_1	X_2	X_2	Y_1	Y_1	Y_2
X_3	X_4	X_4	Y_3	Y_3	Y_4
X_3	X_4	X_4	Y_3	Y_3	Y_4
Z_1	Z_2	Z_2	W_1	W_1	W_2
Z_1	Z_2	Z_2	W_1	W_1	W_2
Z_3	Z_4	Z_4	W_3	W_3	W_4

$\longrightarrow$

X_1	X_2	Y_1	Y_2
X_3	X_4	Y_3	Y_4
Z_1	Z_2	W_1	W_2
Z_3	Z_4	W_3	W_4

$$T_{k-1}(S_1,O_1,X_1,X_2,X_3,X_4) \wedge T_{k-1}(S_2,O_2,X_2,Y_1,X_4,Y_3) \wedge T_{k-1}(S_3,O_3,Y_1,Y_2,Y_3,Y_4) \wedge$$
$$T_{k-1}(S_4,O_4,X_3,X_4,Z_1,Z_2) \wedge T_{k-1}(S_5,O_5,X_4,Y_3,Z_2,W_1) \wedge T_{k-1}(S_6,O_6,Y_3,Y_4,W_1,W_2) \wedge$$
$$T_{k-1}(S_7,O_7,Z_1,Z_2,Z_3,Z_4) \wedge T_{k-1}(S_8,O_8,Z_2,W_1,Z_4,W_3) \wedge T_{k-1}(S_9,O_9,W_1,W_2,W_3,W_4) \rightarrow$$
$$\exists U\ T_k(U,O_1,S_1,S_3,S_7,S_9)$$

(∀-quantifiers are omitted)

NEXPTIME-hardness of ACYCLIC

Inductive step - construct $2^k \times 2^k$ tilings from $2^{k-1} \times 2^{k-1}$ tilings

X_1	X_2	X_2	Y_1	Y_1	Y_2
X_3	X_4	X_4	Y_3	Y_3	Y_4
X_3	X_4	X_4	Y_3	Y_3	Y_4
Z_1	Z_2	Z_2	W_1	W_1	W_2
Z_1	Z_2	Z_2	W_1	W_1	W_2
Z_3	Z_4	Z_4	W_3	W_3	W_4

$\longrightarrow$

X_1	X_2	Y_1	Y_2
X_3	X_4	Y_3	Y_4
Z_1	Z_2	W_1	W_2
Z_3	Z_4	W_3	W_4

$$\forall S \forall O \forall X_1 \forall X_2 \forall X_3 \forall X_4 \, (T_n(S,O,X_1,X_2,X_3,X_4) \rightarrow T(S,O))$$

Concluding NEXPTIME-hardness of ACYCLIC

- Several rules but polynomially many $\Rightarrow$ feasible in polynomial time
- $D \wedge \Sigma \models \exists X\ T(X,t_0)$ iff a $2^n \times 2^n$ tiling exists
- Can be formally shown by induction on *n*

Corollary: BCQ-Answering under ACYCLIC is NEXPTIME-complete w.r.t. the combined complexity

Termination of the Chase

- Drop the existential quantification
    - We obtain the class of full existential rules
    - Very close to Datalog

✓

- Drop the recursive definitions
    - We obtain the class of acyclic existential rules
    - A.k.a. non-recursive existential rules

✓

Sum Up

	Data Complexity	
FULL	PTIME-c	Naïve algorithm
		Reduction from Monotone Circuit Value problem
ACYCLIC	in LOGSPACE	covered later…

	Combined Complexity	
FULL	EXPTIME-c	Naïve algorithm
		Simulation of a deterministic exponential time TM
ACYCLIC	NEXPTIME-c	Small witness property
		Reduction from Tiling problem

Recall our Example

D

person(Alice)

Σ

$\forall X\ (\mathit{Person}(X) \rightarrow \exists Y\ (\mathit{hasParent}(X,Y) \wedge \mathit{Person}(Y)))$

$\text{chase}(D,\Sigma) = D \cup \{\mathit{hasParent}(\text{Alice}, z_1), \mathit{Person}(z_1),$

$\mathit{hasParent}(z_1, z_2), \mathit{Person}(z_2),$

$\mathit{hasParent}(z_2, z_3), \mathit{Person}(z_3), \ldots$

The above rule can be written as the DL-Lite axiom

$\mathit{Person} \sqsubseteq \exists \mathit{hasParent}.\mathit{Person}$

Recall our Example

D

person(Alice)

Σ

$\forall X\ (\mathit{Person}(X) \rightarrow \exists Y\ (\mathit{hasParent}(X,Y) \wedge \mathit{Person}(Y)))$

$\text{chase}(D,\Sigma) = D \cup \{\mathit{hasParent}(\text{Alice}, z_1), \mathit{Person}(z_1),$

$\mathit{hasParent}(z_1, z_2), \mathit{Person}(z_2),$

$\mathit{hasParent}(z_2, z_3), \mathit{Person}(z_3), \ldots$

Existential quantification & recursive definitions

are key features for modelling ontologies

Challenge

We need classes of existential rules such that

- Existential quantification and recursive definition coexist

  $\Rightarrow$ the chase may be infinite

- BCQ-Answering is decidable, and tractable w.r.t. the data complexity

$$\Downarrow$$

Tame the infinite chase:

Deal with infinite structures without explicitly building them

Linear Existential Rules

- A linear existential rule is an existential rule of the form

$$\forall X \forall Y\ (P(X,Y) \rightarrow \exists Z\ \psi(X,Z))$$

  where $P(X,Y)$ is an atom

- We denote LINEAR the class of linear existential rules

- A local property - we can inspect one rule at a time
  $\Rightarrow$ given Σ, we can decide in linear time whether $\Sigma \in$ LINEAR
  $\Rightarrow$ $\Sigma_1 \in$ LINEAR, $\Sigma_2 \in$ LINEAR $\Rightarrow (\Sigma_1 \cup \Sigma_2) \in$ LINEAR

- Strictly more expressive than DL-Lite

LINEAR vs. DL-Lite

Existential rules and DLs rely on first-order semantics - comparable formalisms

DL-Lite Axioms	Existential Rules
$A \sqsubseteq B$	$\forall X\ (A(X) \rightarrow B(X))$
$A \sqsubseteq \exists R$	$\forall X\ (A(X) \rightarrow \exists Y\ R(X,Y))$
$\exists R \sqsubseteq A$	$\forall X \forall Y\ (R(X,Y) \rightarrow A(X))$
$\exists R \sqsubseteq \exists P$	$\forall X \forall Y\ (R(X,Y) \rightarrow \exists Z\ P(X,Z))$
$A \sqsubseteq \exists R.B$	$\forall X\ (A(X) \rightarrow \exists Y\ (R(X,Y) \wedge B(Y)))$
$R \sqsubseteq P$	$\forall X \forall Y\ (R(X,Y) \rightarrow P(X,Y))$
$A \sqsubseteq \neg B$	$\forall X\ (A(X) \wedge B(X) \rightarrow \bot)$

Linear Existential Rules

- A linear existential rule is an existential rule of the form

$$\forall X \forall Y\ (P(X,Y) \rightarrow \exists Z\ \psi(X,Z))$$

  where $P(X,Y)$ is an atom

- We denote LINEAR the class of linear existential rules

- A local property - we can inspect one rule at a time
  $\Rightarrow$ given Σ, we can decide in linear time whether $\Sigma \in$ LINEAR
  $\Rightarrow$ $\Sigma_1 \in$ LINEAR, $\Sigma_2 \in$ LINEAR $\Rightarrow$ $(\Sigma_1 \cup \Sigma_2) \in$ LINEAR

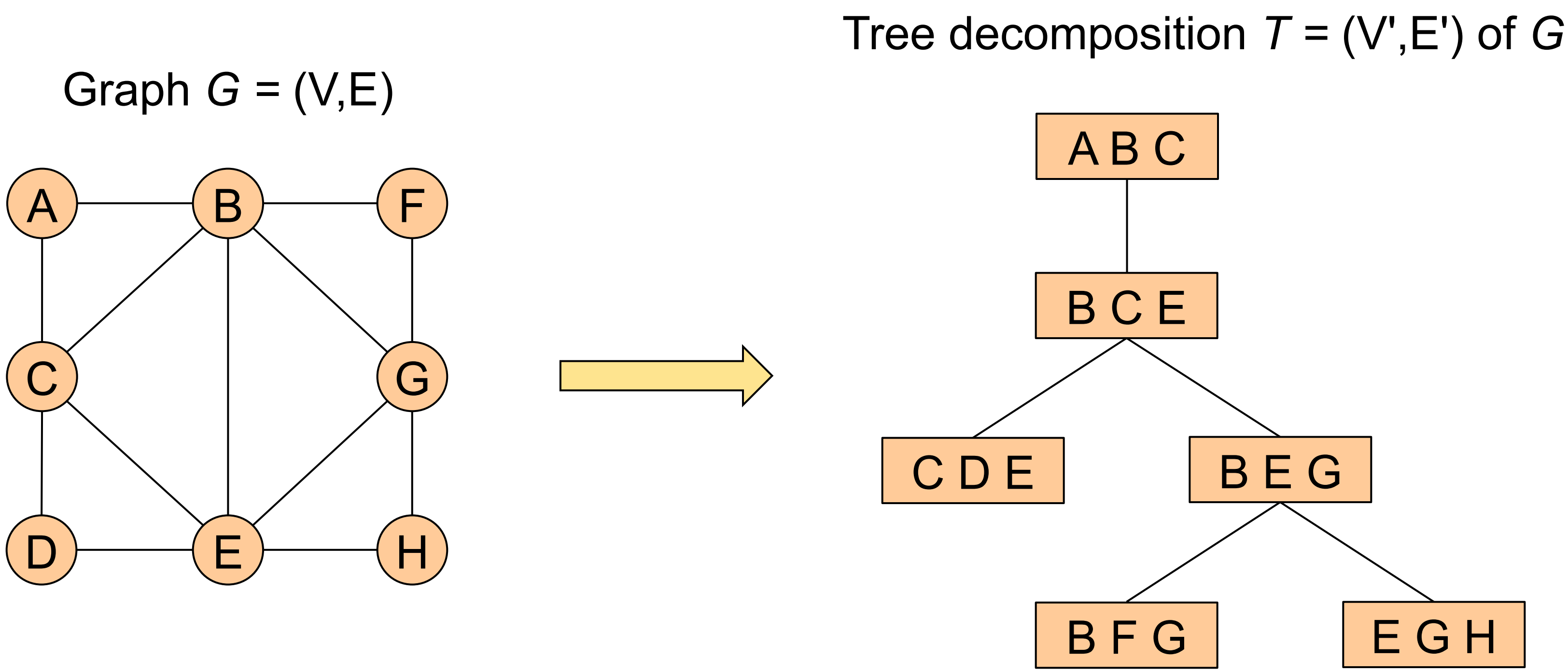
- Strictly more expressive than DL-Lite

- Infinite chase - $\forall X\ (\mathit{Person}(X) \rightarrow \exists Y\ (\mathit{hasParent}(X,Y) \wedge \mathit{Person}(Y)))$

- But, BCQ-Answering is decidable - the chase has finite treewidth

Treewidth of a Graph

Tree decomposition - a mapping of a graph into a tree



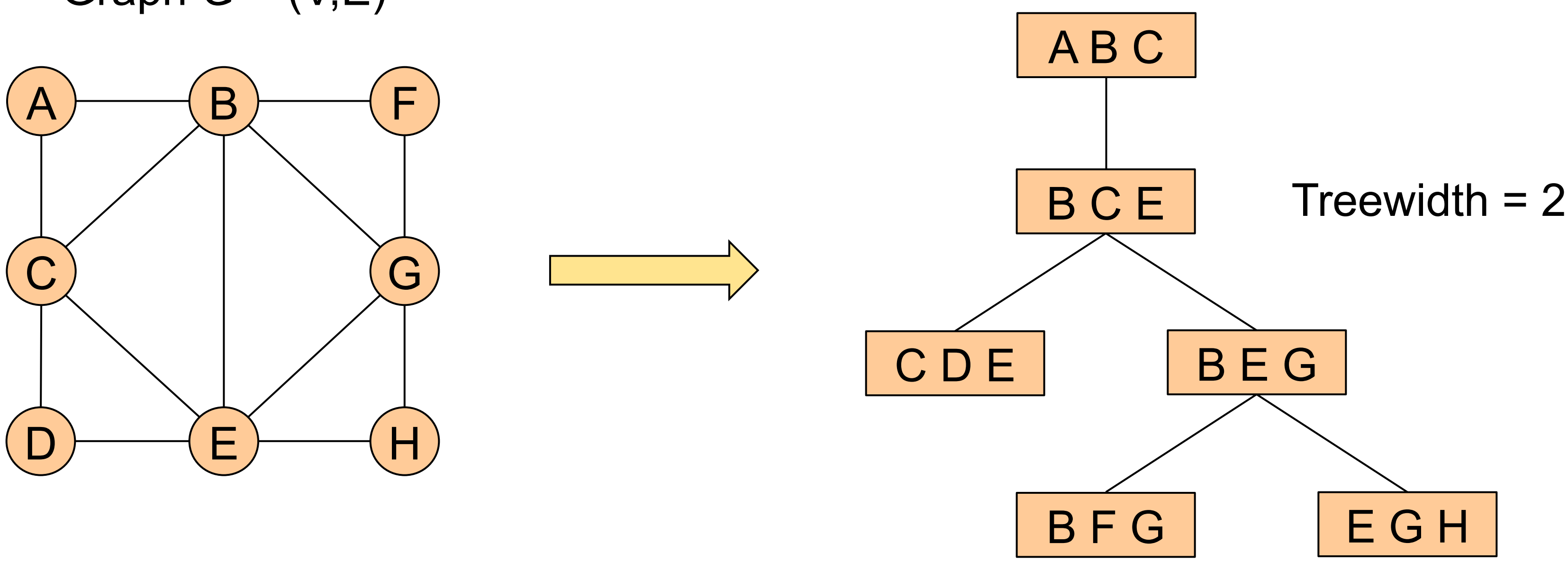
1. For each v ∈ V, there exists u ∈ V' such that v ∈ u
2. For each (v,w) ∈ E, there exists u ∈ V' such that {v,w} ⊆ u
3. For each v ∈ V, {u | v ∈ u} induces a connected subtree

Treewidth of a Graph

Tree decomposition - a mapping of a graph into a tree

Tree decomposition T = (V',E') of G

Graph G = (V,E)



- The width of T is defined as $\max_{u \in V'} \{|u|\} - 1$
- The treewidth of G, denoted tw(G), is the minimum width over all tree decompositions of G

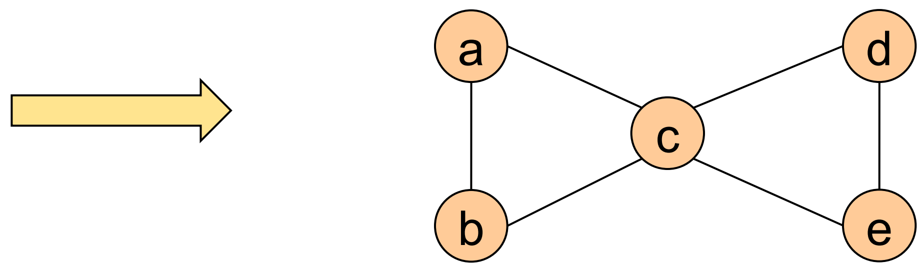
Treewidth of an Instance

- An instance *J* can be represented as a graph $\mathcal{G}_J$ - Gaifman graph

R(a,b,c)

S(c,d)

T(c,d,e)



- The treewidth of *J*, denoted tw(*J*), is defined as tw($\mathcal{G}_J$)

- Thus, we can talk about the treewidth of the chase

- Lemma: For a database *D*, and a set $\Sigma \in$ LINEAR, tw(chase(*D*,Σ)) is finite

Decidability of LINEAR

Theorem: BCQ-Answering under LINEAR is decidable

Proof: The ingredients of the proof are the following:

1. The chase under LINEAR has finite treewidth
2. The tree model property implies decidability of satisfiability - classical result

A fragment $\mathcal{L}$ of first-order logic enjoys the tree model property if: for every $\varphi \in \mathcal{L}$,

if φ is satisfiable, then there exists $J \in$ models(φ) such that tw(J) is finite

Decidability of LINEAR

Theorem: BCQ-Answering under LINEAR is decidable

Proof: The ingredients of the proof are the following:

1. The chase under LINEAR has finite treewidth
2. The tree model property implies decidability of satisfiability - classical result

- Consider a database D, a set $\Sigma \in$ LINEAR, and a BCQ Q
- Clearly, $D \wedge \Sigma \vDash Q$ iff $D \wedge \Sigma \wedge \neg Q \vDash \bot$
- Thus, it suffices to show that, if $D \wedge \Sigma \wedge \neg Q$ is satisfiable, then it has a model of finite treewidth
- By universality, $D \wedge \Sigma \wedge \neg Q$ is satisfiable implies chase(D,Σ) $\wedge \neg Q$ is satisfiable
- Therefore, $D \wedge \Sigma \wedge \neg Q$ is satisfiable implies chase(D,Σ) is a model of $D \wedge \Sigma \wedge \neg Q$
- The claim follows since tw(chase(D,Σ)) is finite

Decidability of LINEAR

Theorem: BCQ-Answering under LINEAR is decidable

Proof: The ingredients of the proof are the following:

1. The chase under LINEAR has finite treewidth
2. The tree model property implies decidability of satisfiability - classical result

…but, what about the complexity of the problem?

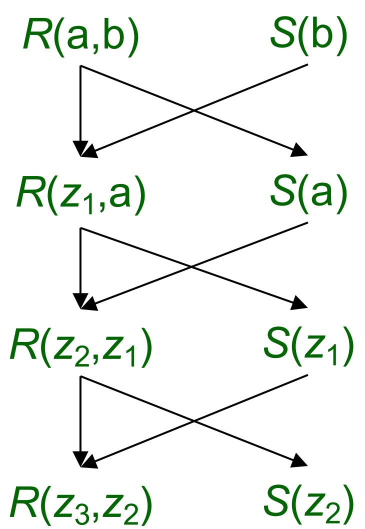
we need new techniques

Chase Graph

The chase can be naturally seen as a graph - chase graph

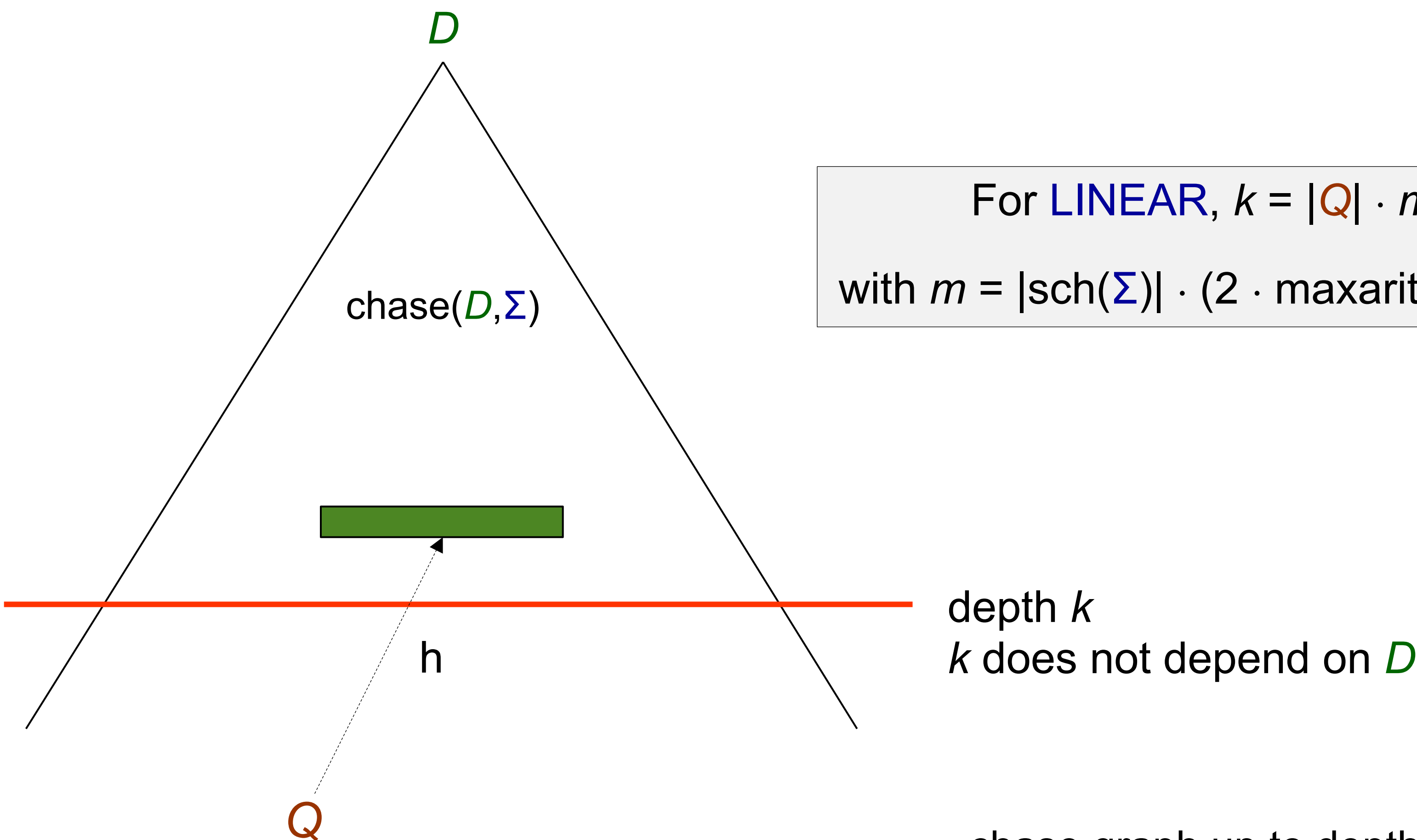
$D = \{R(a,b), S(b)\}$

$$\Sigma = \begin{cases} \forall X \forall Y\ (R(X,Y) \wedge S(Y) \rightarrow \exists Z\ R(Z,X)) \\ \forall X \forall Y\ (R(X,Y) \rightarrow S(X)) \end{cases}$$



For LINEAR, the chase graph is a forest

Bounded Derivation-Depth Property



For LINEAR, $k = |Q| \cdot m$

with $m = |\mathrm{sch}(\Sigma)| \cdot (2 \cdot \mathrm{maxarity})^{\mathrm{maxarity}}$

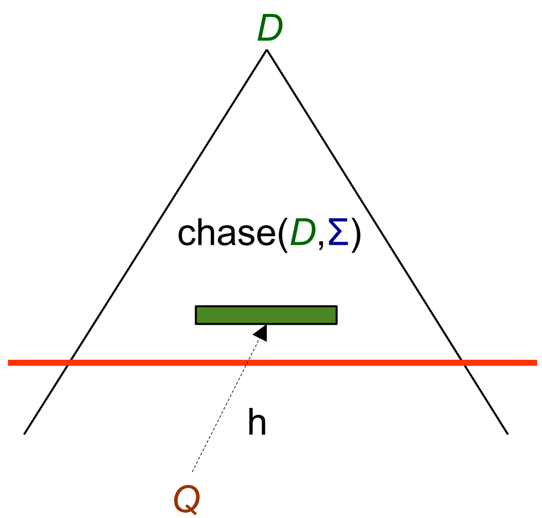
chase graph up to depth k

$$\mathrm{chase}(D,\Sigma) \vDash Q \quad \Rightarrow \quad \mathrm{chase}^k(D,\Sigma) \vDash Q$$

The Blocking Algorithm for LINEAR

- The blocking algorithm shows that BCQ-Answering under **LINEAR** is
  - in PTIME w.r.t. the data complexity
  - in 2EXPTIME w.r.t. the combined complexity

…we can do better than the blocking algorithm



$k = |Q| \cdot |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$

Data Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in LOGSPACE w.r.t. the data complexity

Proof: Not so easy! Different techniques must be applied. This will be the subject of the second part of our course.

Combined Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in NEXPTIME w.r.t. the combined complexity

Proof: We first need to establish the so-called small witness property

Small Witness Property for LINEAR

Lemma: chase(D,Σ) $\models Q$ $\Rightarrow$ there exists a chase sequence

$$D\langle\sigma_1,h_1\rangle J_1\langle\sigma_2,h_2\rangle J_2\langle\sigma_3,h_3\rangle J_3 \ldots \langle\sigma_n,h_n\rangle J_n$$

of D w.r.t. Σ with

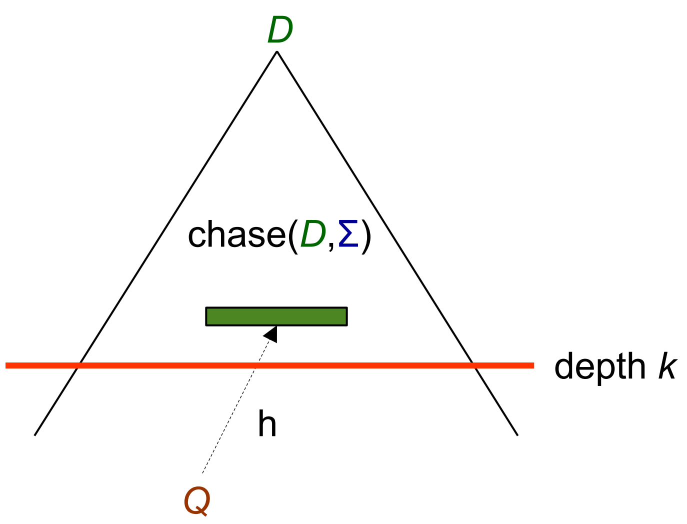
$$n = (|Q|)^2 \cdot |\text{sch}(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$$

such that $J_n \models Q$

Proof:

- By hypothesis, there exists a homomorphism h such that h(Q) $\subseteq$ chase(D, Σ)
- By the bounded-derivation depth property

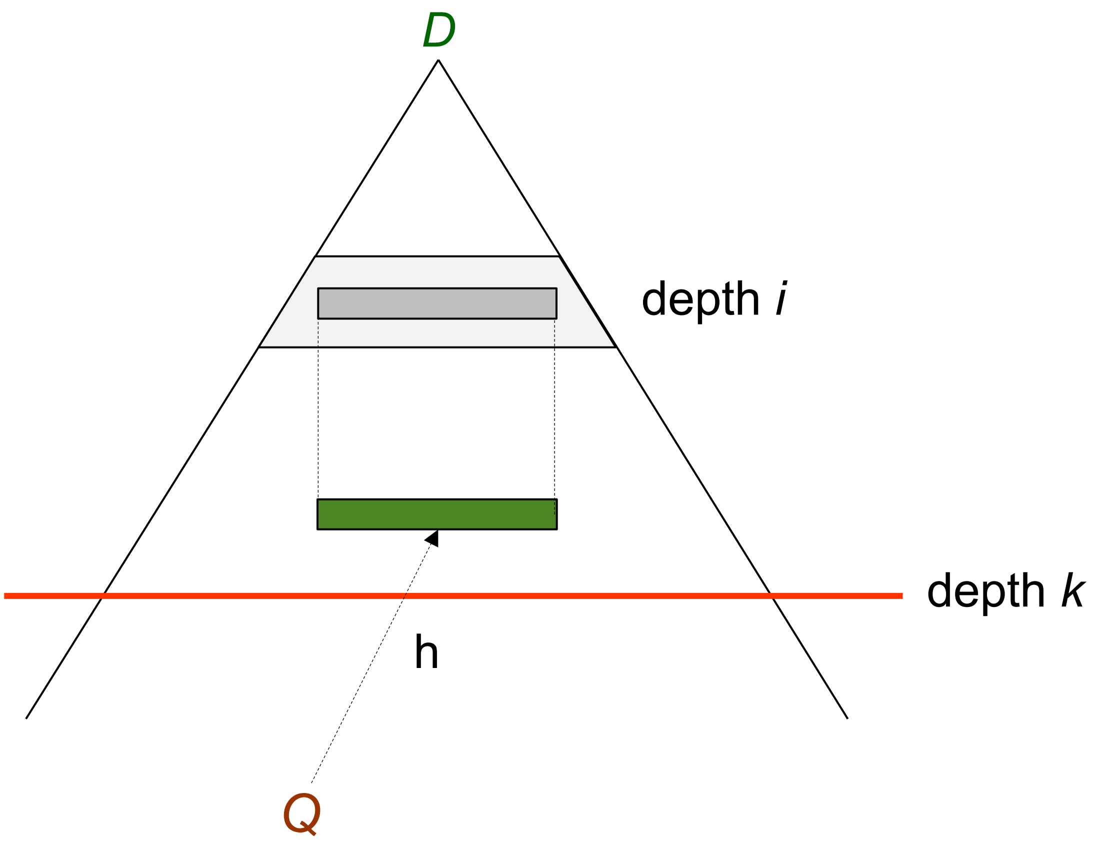
$$k = |Q| \cdot |\text{sch}(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$$



Small Witness Property for LINEAR

Proof (cont.):

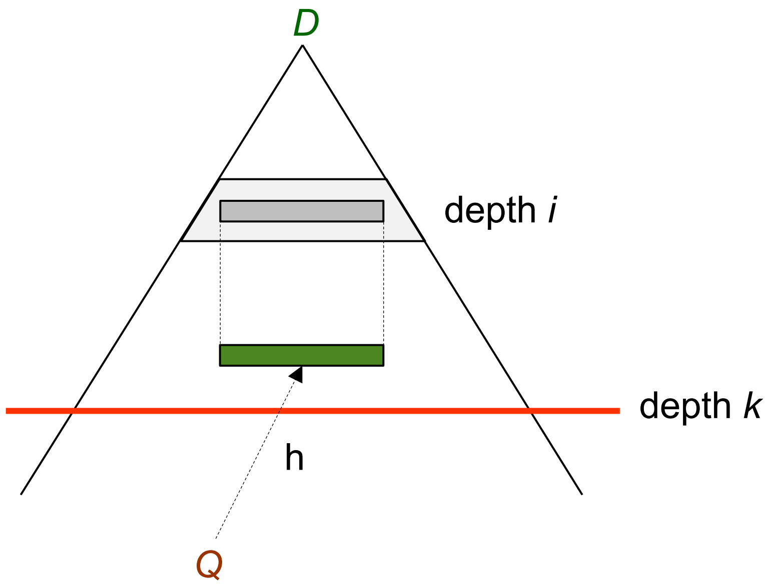
- Let us focus on depth *i* of the chase graph
- How many atoms do we need?
- No more than |*Q*|



Small Witness Property for LINEAR

Proof (cont.):

- Let us focus on depth *i* of the chase graph
- How many atoms do we need?
- No more than $|Q|$
- Thus, to entail the query we need at most



$$k \cdot |Q|$$

$$= |Q| \cdot |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}} \cdot |Q|$$

$$= (|Q|)^2 \cdot |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$$

Combined Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in NEXPTIME w.r.t. the combined complexity

Proof: Consider a database D, a set $\Sigma \in$ LINEAR, and a BCQ Q

Having the small witness property in place, we can exploit the following algorithm:

1. Non-deterministically construct a chase sequence

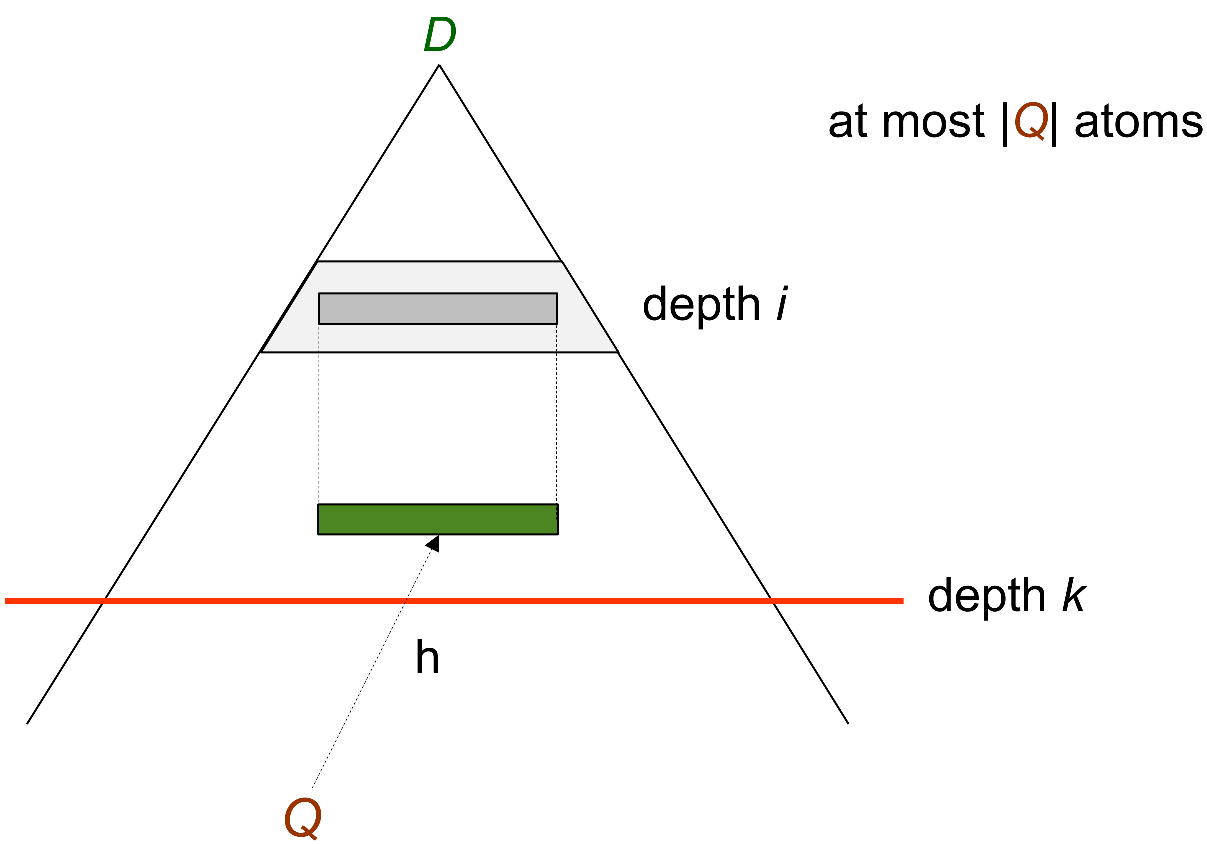
$$D\langle\sigma_1,h_1\rangle J_1\langle\sigma_2,h_2\rangle J_2\langle\sigma_3,h_3\rangle J_3 \ldots \langle\sigma_n,h_n\rangle J_n$$

   of D w.r.t. Σ with $n = (|Q|)^2 \cdot |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$

2. Check for the existence of a homomorphism h such that $h(Q) \subseteq J_n$

Can we do better? Any ideas?

Key Observation

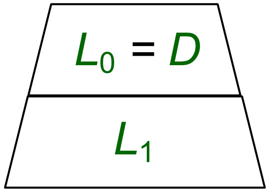


level-by-level construction

Combined Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in PSPACE w.r.t. the combined complexity

Proof:



Combined Complexity of LINEAR

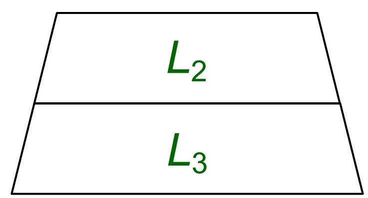
Theorem: BCQ-Answering under LINEAR is in PSPACE w.r.t. the combined complexity

Proof:

Combined Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in PSPACE w.r.t. the combined complexity

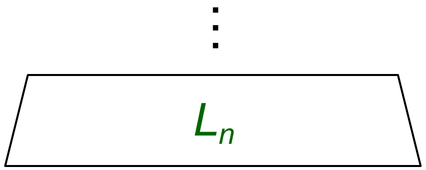
Proof:



Combined Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in PSPACE w.r.t. the combined complexity

Proof:



Combined Complexity of LINEAR

Theorem: BCQ-Answering under LINEAR is in PSPACE w.r.t. the combined complexity

Proof (cont.):

At each step we need to maintain

- $O(|Q|)$ atoms
- A counter $ctr \leq (|Q|)^2 \cdot |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$

- Thus, we need polynomial space

- The claim follows since NPSPACE = PSPACE

Combined Complexity of LINEAR

We cannot do better than the previous algorithm

Theorem: BCQ-Answering under LINEAR is PSPACE-hard w.r.t. the combined complexity

Proof : By simulating a deterministic polynomial space Turing machine

PSPACE-hardness of LINEAR

Our Goal: Encode the polynomial space computation of a DTM M on input string I using a database D, a set $\Sigma \in$ LINEAR, and a BCQ Q such that

$D \wedge \Sigma \vDash Q$ iff M accepts I using at most $n = (|I|)^k$ cells

PSPACE-hardness of LINEAR

- Assume that the tape alphabet is $\{0,1,\sqcup\}$
- Suppose that M halts on $I = \alpha_1 \ldots \alpha_m$ using $n = m^k$ cells, for $k > 0$

Initial configuration - the database D

$$\mathit{Config}(s_{init},\alpha_1,\ldots,\alpha_m,\underbrace{\sqcup,\ldots,\sqcup}_{n-m},1,\underbrace{0,\ldots,0}_{n-1})$$

PSPACE-hardness of LINEAR

- Assume that the tape alphabet is $\{0,1,\sqcup\}$
- Suppose that M halts on $I = \alpha_1 \ldots \alpha_m$ using $n = m^k$ cells, for $k > 0$

Transition rule - $\delta(s_1,\alpha) = (s_2,\beta,+1)$

for each $i \in \{1,\ldots,n\}$:

$$\forall X\ (\mathit{Config}(s_1,X_1,\ldots,X_{i-1},\alpha,X_{i+1},\ldots,X_n,\overbrace{0,\ldots,0}^{i-1},1,\overbrace{0,\ldots,0}^{n-i}) \rightarrow$$

$$\mathit{Config}(s_2,X_1,\ldots,X_{i-1},\beta,X_{i+1},\ldots,X_n,\underbrace{0,\ldots,0}_{i},1,\underbrace{0,\ldots,0}_{n-i-1}))$$

PSPACE-hardness of LINEAR

- Assume that the tape alphabet is {0,1,⊔}
- Suppose that *M* halts on $I = \alpha_1 \ldots \alpha_m$ using $n = m^k$ cells, for $k > 0$

$$D \wedge \Sigma \models \exists X \, Config(s_{acc}, X) \quad \text{iff} \quad M \text{ accepts } I$$

…but, the rules are not constant-free

we can eliminate the constants by applying a simple trick

PSPACE-hardness of LINEAR

Initial configuration - the database *D*

auxiliary constants for the states and the tape alphabet

$$\mathit{Config}(s_{init},\alpha_1,\ldots,\alpha_m,\underbrace{\sqcup,\ldots,\sqcup}_{n-m},1,\underbrace{0,\ldots,0}_{n-1},s_1,\ldots s_\ell,0,1,\sqcup)$$

PSPACE-hardness of LINEAR

Transition rule - $\delta(s_1,0) = (s_2,\sqcup,+1)$

for each $i \in \{1,\ldots,n\}$:

$$Config(S_1,X_1,\ldots,X_{i-1},Z,X_{i+1},\ldots,X_n,\underbrace{Z,\ldots,Z}_{i-1},O,\underbrace{Z,\ldots,Z}_{n-i},S_1,\ldots S_\ell,Z,O,B) \rightarrow$$

$$Config(S_2,X_1,\ldots,X_{i-1},B,X_{i+1},\ldots,X_n,\underbrace{Z,\ldots,Z}_{i},O,\underbrace{Z,\ldots,Z}_{n-i-1},S_1,\ldots S_\ell,Z,O,B)$$

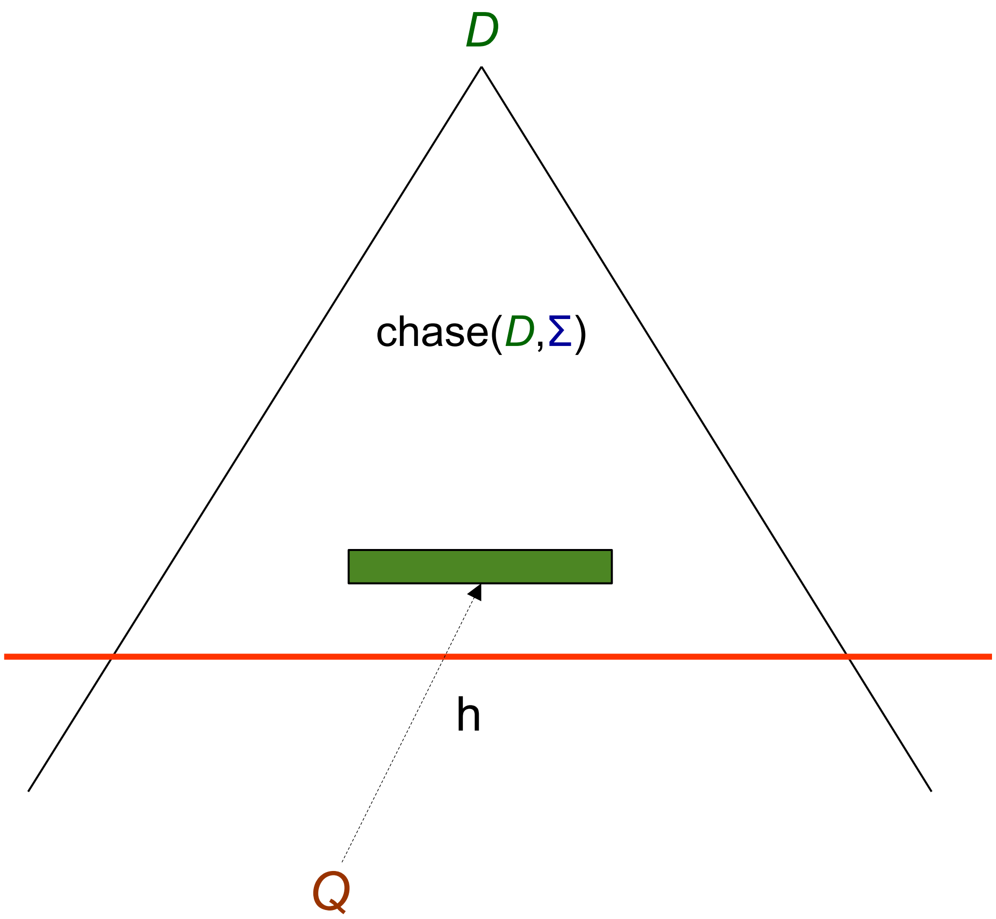
(∀-quantifiers are omitted)

Sum Up

	Data Complexity	
FULL	PTIME-c	Naïve algorithm
		Reduction from Monotone Circuit Value problem
ACYCLIC	in LOGSPACE	**Second part of our course**
LINEAR		

	Combined Complexity	
FULL	EXPTIME-c	Naïve algorithm
		Simulation of a deterministic exponential time TM
ACYCLIC	NEXPTIME-c	Small witness property
		Reduction from Tiling problem
LINEAR	PSPACE-c	Level-by-level non-deterministic algorithm
		Simulation of a deterministic polynomial space TM

Forward Chaining Techniques



Useful techniques for establishing optimal upper bounds

…but not practical - we need to store instances of very large size