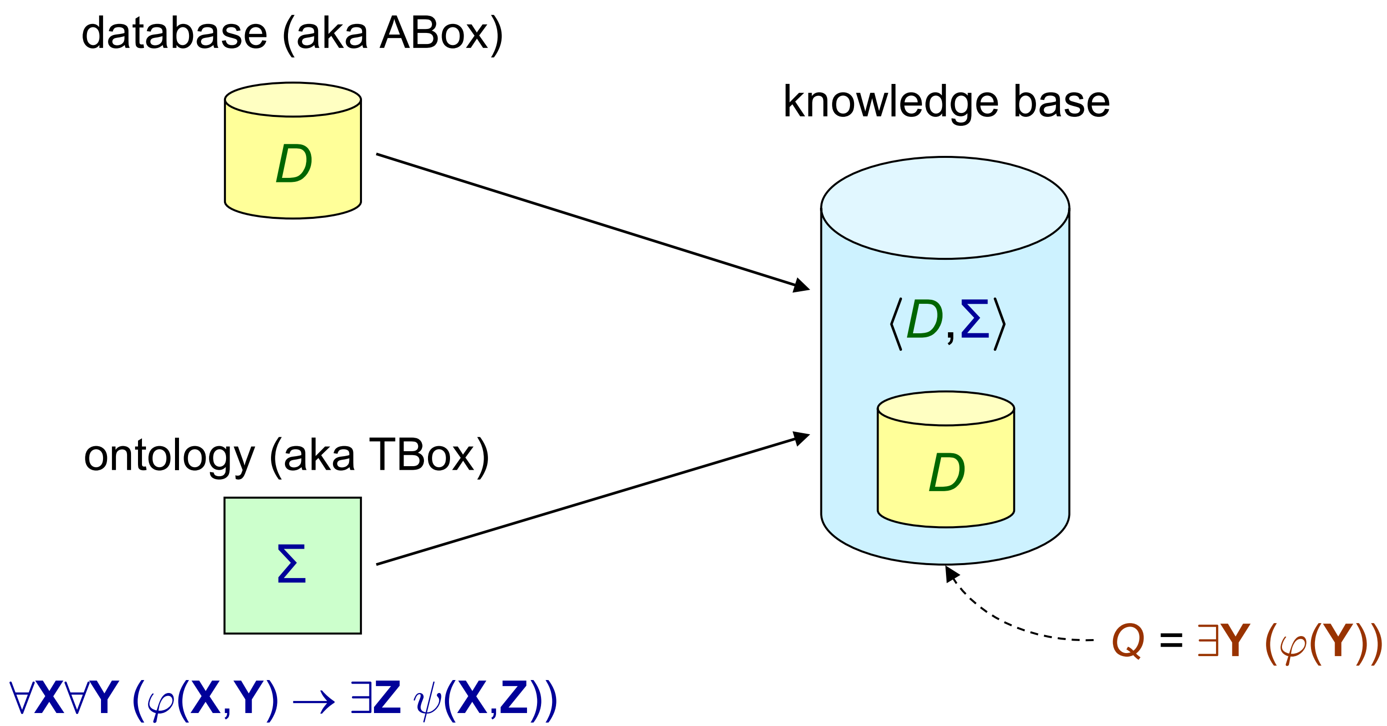
Sebastian Rudolph
International Center for Computational Logic
TU Dresden

Existential Rules – Lecture 5

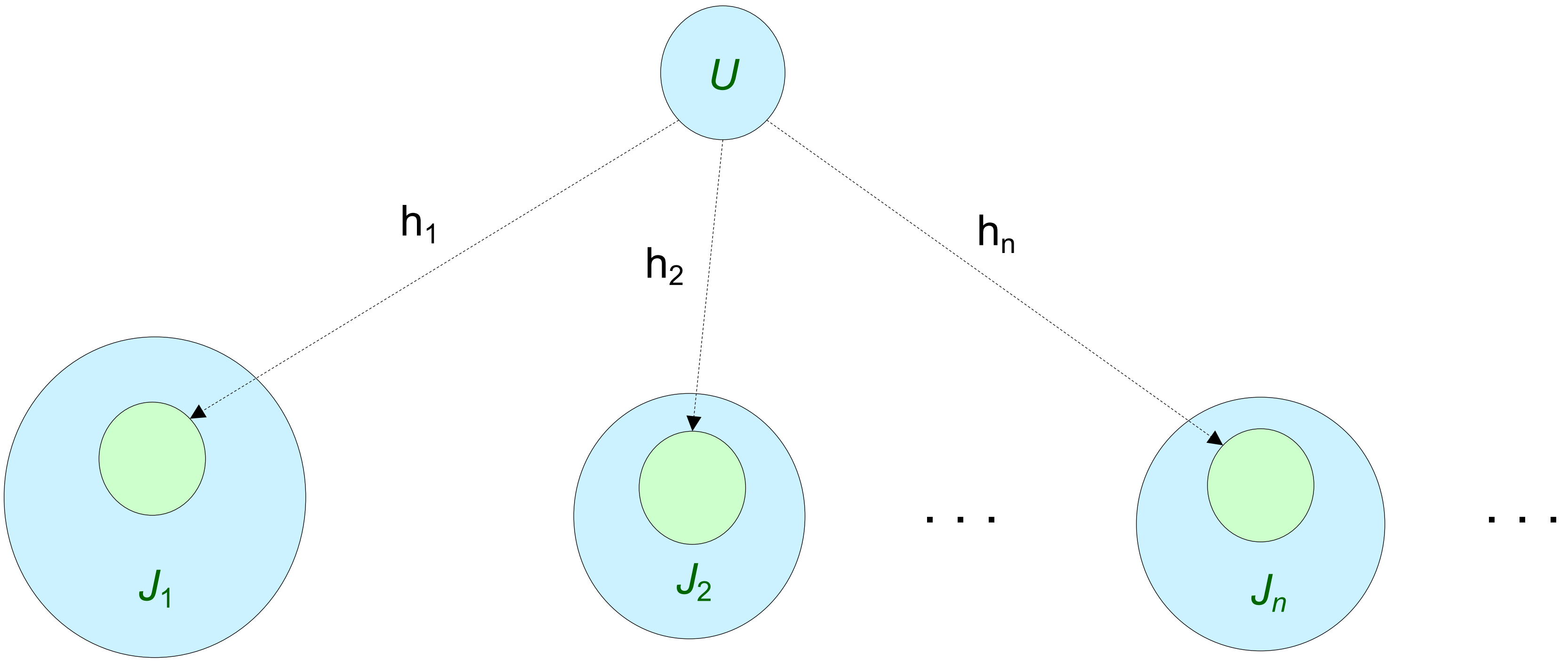
Adapted from slides by Andreas Pieris and Michaël Thomazo
Summer Term 2023

BCQ-Answering: Our Main Decision Problem



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

Universal Models (a.k.a. Canonical Models)



An instance U is a universal model of $D \wedge \Sigma$ if the following holds:

1. U is a model of $D \wedge \Sigma$

2. $\forall J \in \text{models}(D \wedge \Sigma)$, there exists a homomorphism h_J such that $h_J(U) \subseteq J$

Query Answering via the Chase

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

+

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

=

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

Rest of the Lectrure

- Undecidability of BCQ-Answering
- Gaining decidability - terminating chase
- Full Existential Rules
- Acyclic Existential Rules

Undecidability of BCQ-Answering

Theorem: BCQ-Answering is undecidable

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

…syntactic restrictions are needed!!!

What is the Source of Non-termination?

D

person(Alice)

Σ

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

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

$hasParent(z_1, z_2), Person(z_2),$

$hasParent(z_2, z_3), Person(z_3), \ldots$

1. Existential quantification

2. Recursive definitions

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

Full Existential Rules

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

$$\forall X \forall Y\ (\varphi(X,Y) \rightarrow \psi(X))$$

- We denote FULL the class of full existential rules

- A local property - we can inspect one rule at a time

  $\Rightarrow$ given Σ, we can decide in linear time whether $\Sigma \in$ FULL

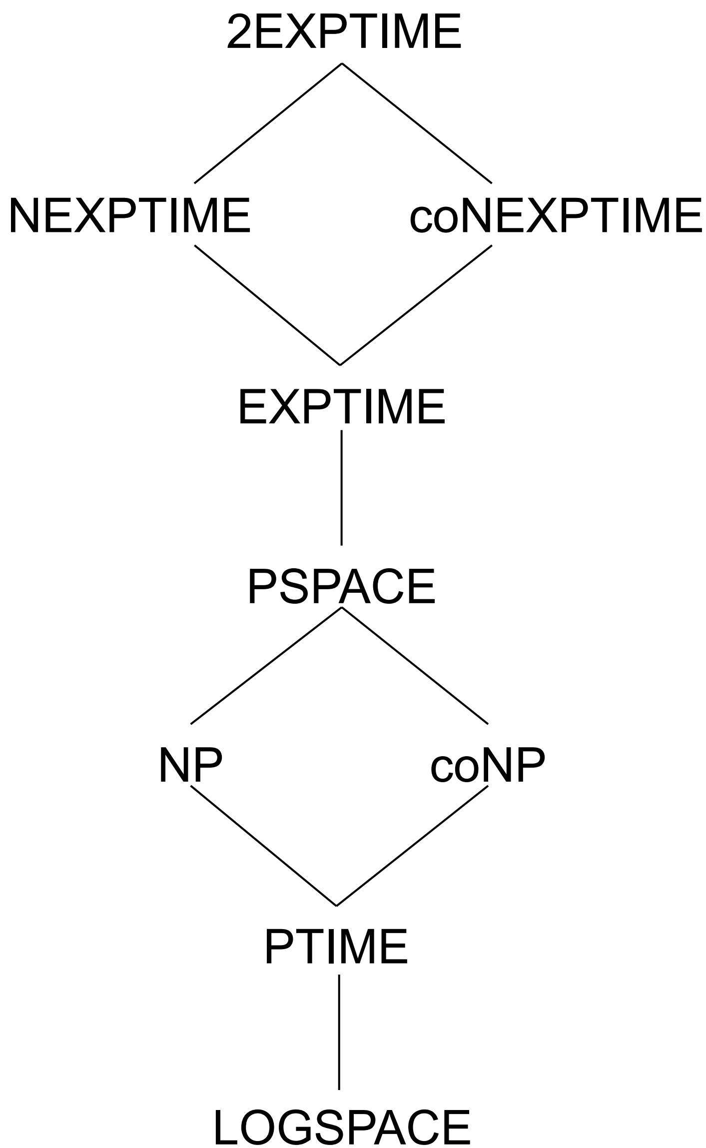
  $\Rightarrow$ closed under union - $\Sigma_1 \in$ FULL, $\Sigma_2 \in$ FULL $\Rightarrow (\Sigma_1 \cup \Sigma_2) \in$ FULL

- Why does the chase terminate?

Complexity Measures for Query Answering

- Data complexity: is calculated by considering only the database as part of the input, while the ontology and the query are fixed
- Combined complexity: is calculated by considering, apart from the database, also the ontology and the query as part of the input
- Data complexity vs. Combined complexity
  - Data complexity tends to be a more meaningful measure - ontologies and queries tend to be small; databases tend to be large
  - Nevertheless, the combined complexity is a relevant measure - identifies the real source of complexity

Some Important Complexity Classes



Problems that can be solved by an algorithm that runs in double-exponential time

We need the power of non-determinism

Problems that can be solved by an algorithm that runs in exponential time

Problems that can be solved by an algorithm that uses a polynomial amount of memory

We need the power of non-determinism

Problems that can be solved by an algorithm that runs in polynomial time

Problems that can be solved by an algorithm that uses a logarithmic amount of memory

Data Complexity of FULL

Theorem: BCQ-Answering under FULL is in PTIME w.r.t. the data complexity

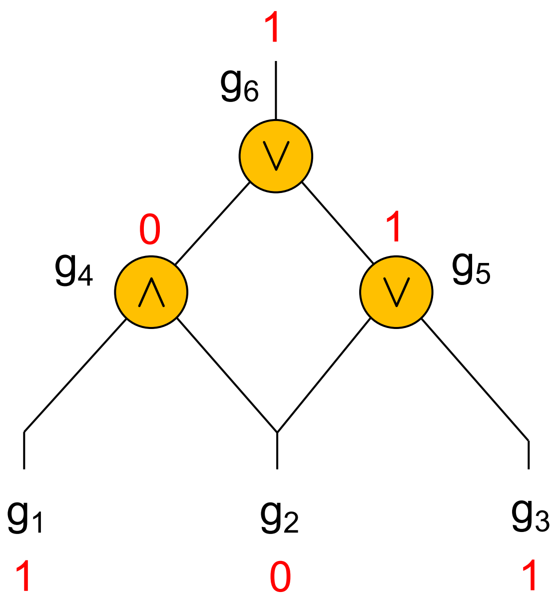
(Analysis of “brute force” materialization and querying algorithm.)

We cannot do better than the naïve algorithm

Theorem: BCQ-Answering under FULL is PTIME-hard w.r.t. the data complexity

Proof : By a LOGSPACE reduction from Monotone Circuit Value problem

Data Complexity of FULL



Does the circuit evaluate to *true*?

encoding of the circuit as a database *D*

$T(g_1)$ $T(g_3)$

$AND(g_4,g_1,g_2)$ $OR(g_5,g_2,g_3)$ $OR(g_6,g_4,g_5)$

evaluation of the circuit via a *fixed* set Σ

$\forall X \forall Y \forall Z\, (T(X) \wedge OR(Z,X,Y) \rightarrow T(Z))$

$\forall X \forall Y \forall Z\, (T(Y) \wedge OR(Z,X,Y) \rightarrow T(Z))$

$\forall X \forall Y \forall Z\, (T(X) \wedge T(Y) \wedge AND(Z,X,Y) \rightarrow T(Z))$

Circuit evaluates to *true* iff $D \wedge \Sigma \vDash T(g_6)$

Combined Complexity of FULL

Theorem: BCQ-Answering under FULL is in EXPTIME w.r.t. the combined complexity

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

We apply the naïve algorithm:

1. Construct chase(D,Σ)
2. Check for the existence of a homomorphism h such that h(Q) $\subseteq$ chase(D,Σ)

By our previous analysis, in the worst case, the naïve algorithm runs in time

$$(|\mathrm{sch}(\Sigma)| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxarity}})^2 \cdot |\Sigma| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxvariables}(\Sigma)} \cdot \mathrm{maxbody}(\Sigma)$$

$$+$$

$$(|\mathrm{adom}(D)|)^{\#\mathrm{variables}(Q)} \cdot |Q| \cdot |\mathrm{sch}(\Sigma)| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxarity}}$$

Combined Complexity of FULL

We cannot do better than the naïve algorithm

Theorem: BCQ-Answering under FULL is EXPTIME-hard w.r.t. the combined complexity

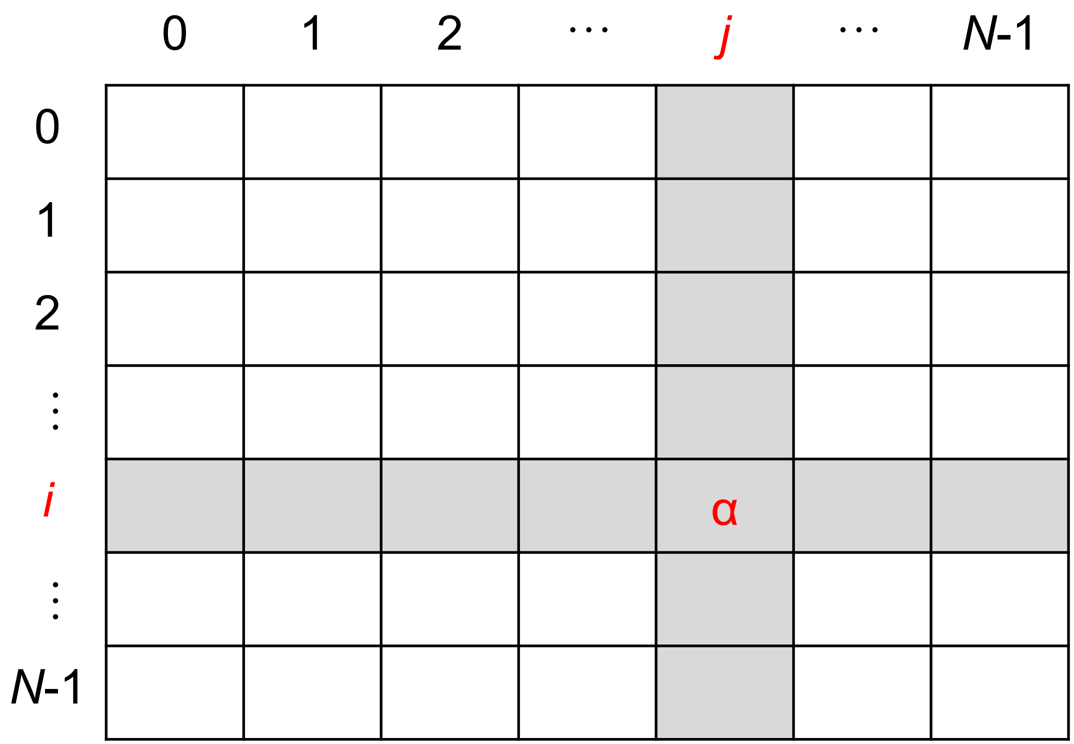
Proof : By simulating a deterministic exponential time Turing machine

EXPTIME-hardness of FULL

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

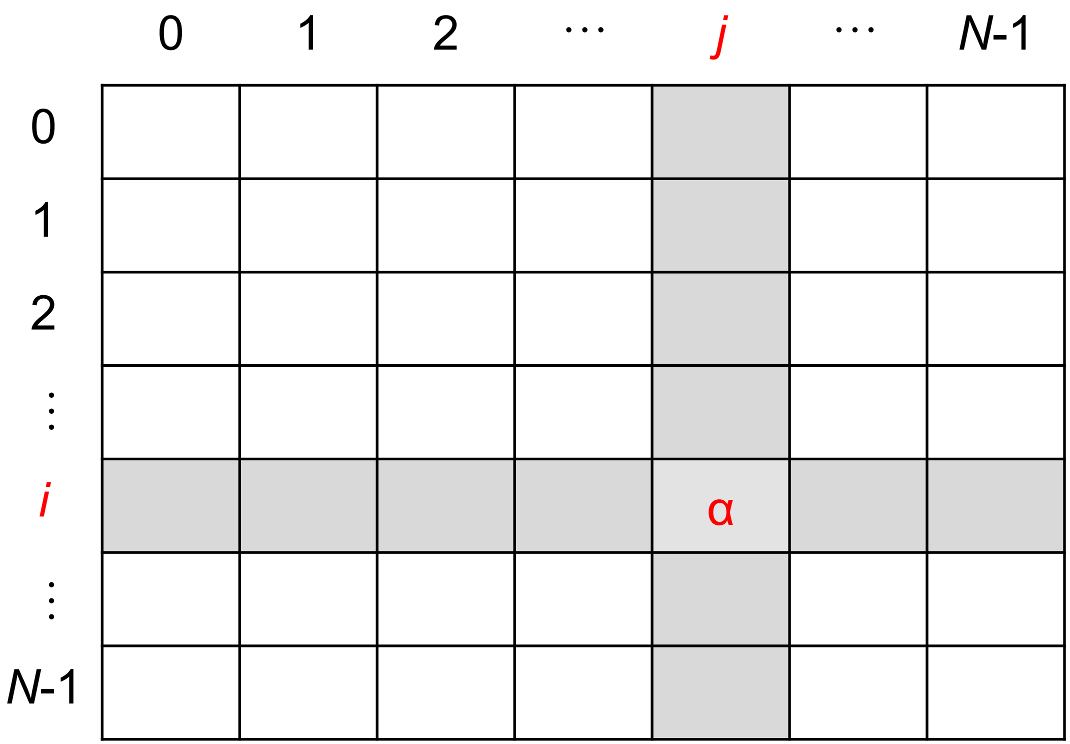
$$D \wedge \Sigma \vDash Q \text{ iff } M \text{ accepts } I \text{ in at most } N = 2^m \text{ steps, where } m = |I|^k$$

The Schema



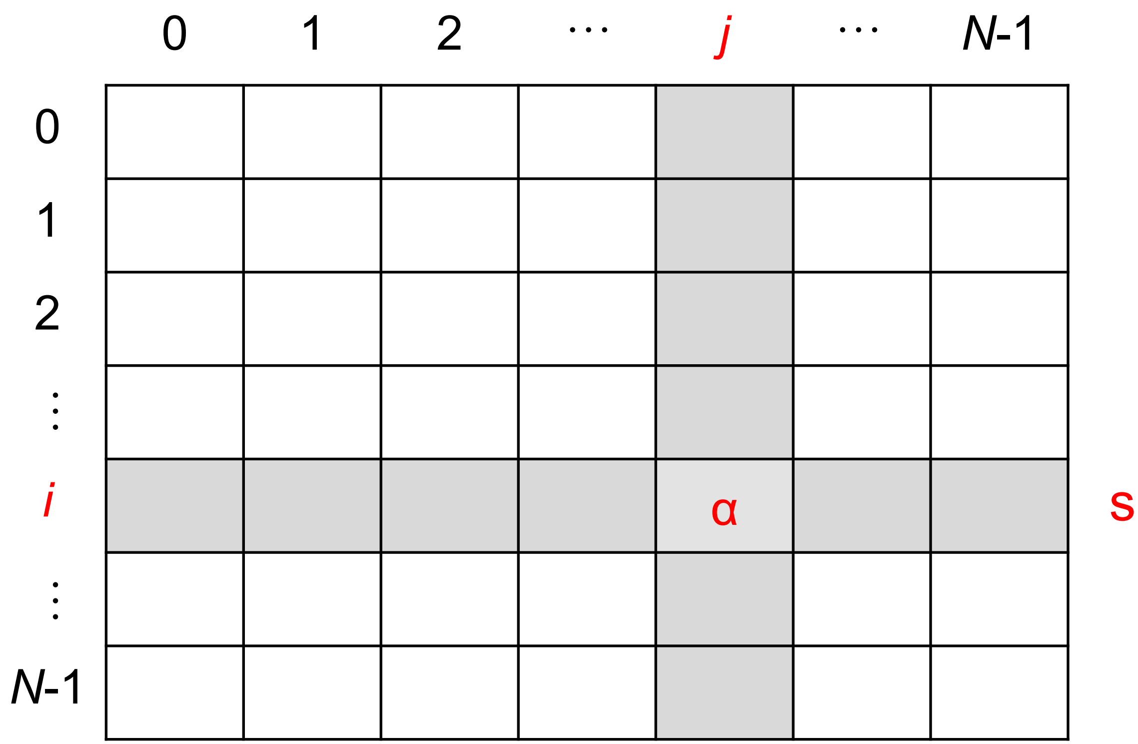
Symbol[α](*i*,*j*) - at time instant *i*, cell *j* contains α

The Schema



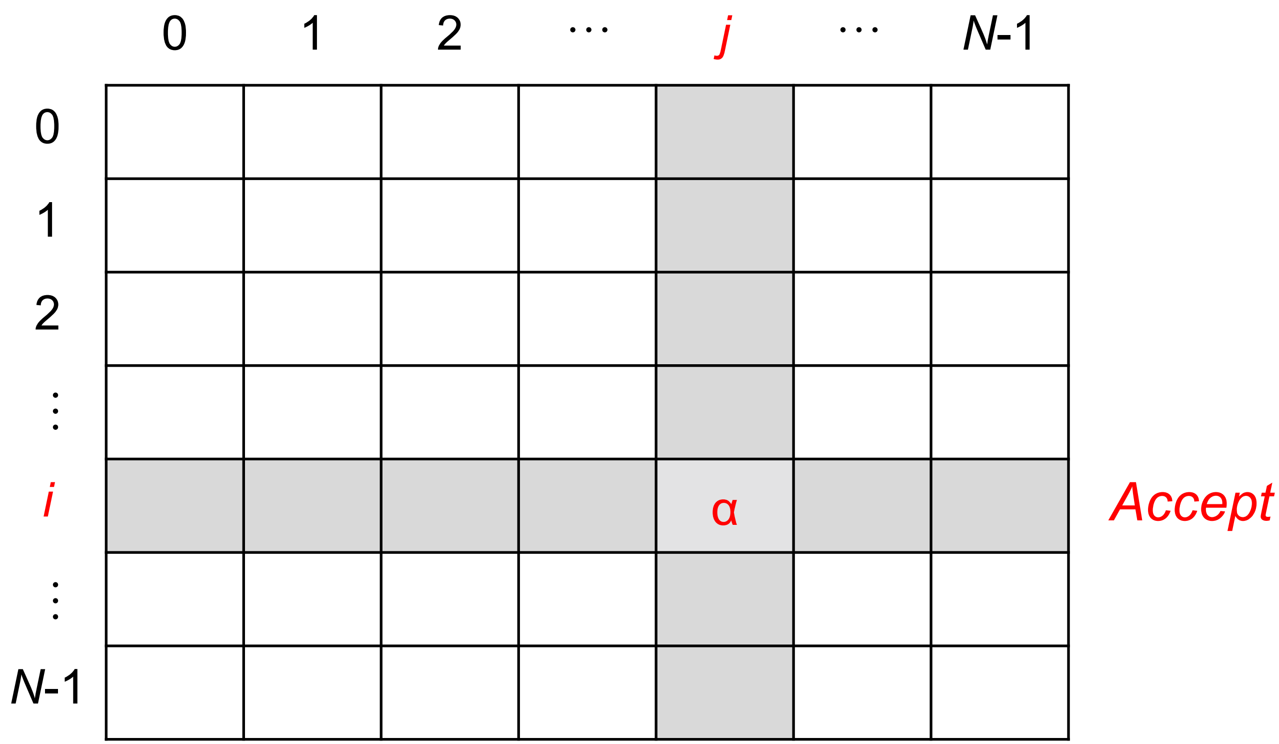
Cursor(i,j) - at time instant *i*, cursor points to cell *j*

The Schema



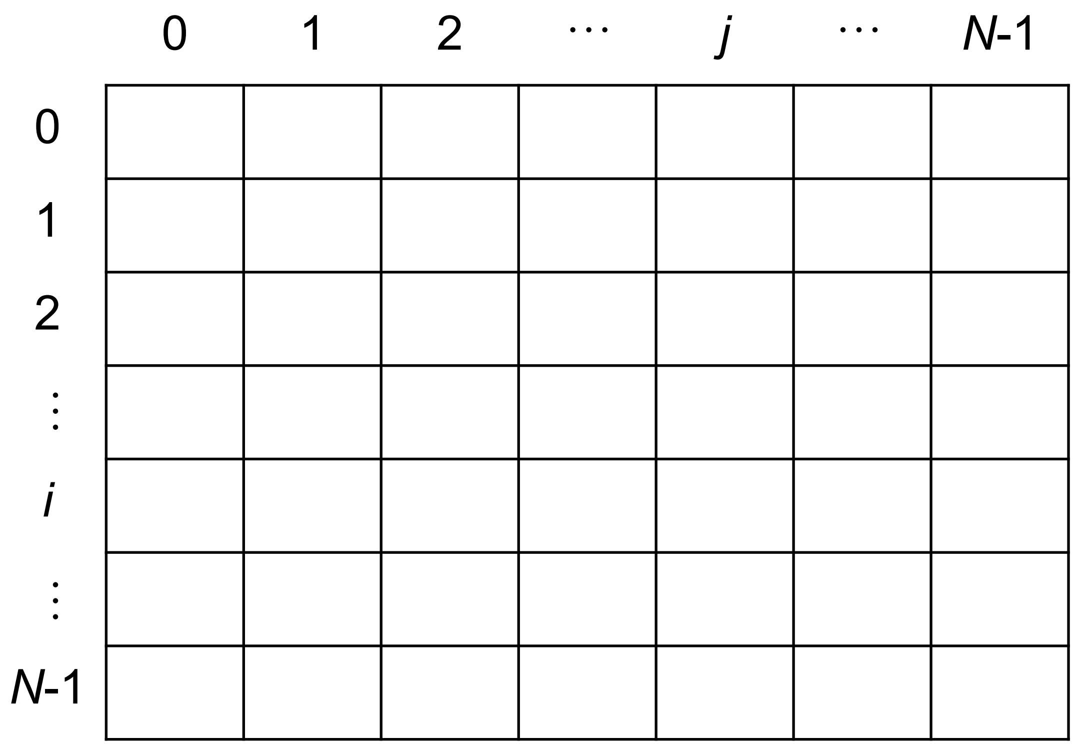
State[s](*i*) - at time instant *i*, the machine is in state *s*

The Schema



Accept(i) - at time instant *i*, the machine accepts

The Schema



First(0), *Succ*(0,1), *Succ*(1,2), *Succ*(2,3), …, *Succ*(*N*-2,*N*-1)

$\prec$ - transitive closure of *Succ*

} will be defined later

Initialization Rules

Assume that $I = \alpha_0 \ldots \alpha_{n-1}$

	0	1	2	⋯	n-1	n	⋯	N-1	
0	α_0	α_1	α_2	…	α_{n-1}	$\sqcup$	…	$\sqcup$	s_0

$$\forall T\, (\mathit{First}(T) \rightarrow \mathit{Symbol}[\alpha_i](T,i) \wedge \mathit{Cursor}(T,T) \wedge \mathit{State}[s_0](T))$$

$$\forall T \forall C\, (\mathit{First}(T) \wedge \prec(n\text{-}1,C) \rightarrow \mathit{Symbol}[\sqcup](T,C))$$

Transition Rules

$\delta(s_1,\alpha) \;=\; (s_2,\beta,+1)$

	j	j+1	j+2	
i	x	α	y	s_1
i+1	x	β	y	s_2

$$\forall T \forall T_1 \forall C \forall C_1 \; (\mathit{State}[s_1](T) \wedge \mathit{Cursor}(T,C) \wedge \mathit{Symbol}[\alpha](T,C) \wedge \mathit{Succ}(T,T_1) \wedge \mathit{Succ}(C,C_1) \rightarrow$$

$$\mathit{Symbol}[\beta](T_1,C) \wedge \mathit{Cursor}(T_1,C_1) \wedge \mathit{State}[s_2](T_1))$$

Inertia Rules

Cells that are not changed during the transition keep their old values

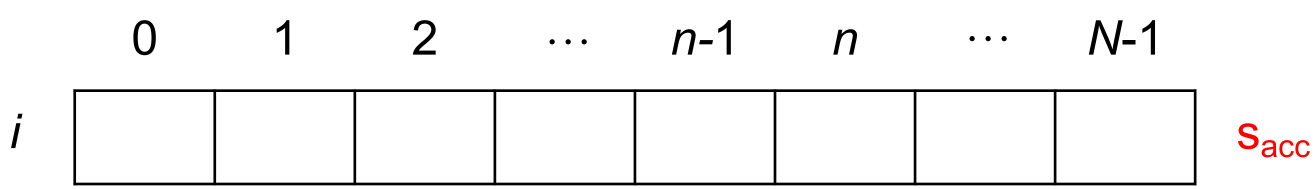
	j	j+1	j+2	
i	x	α	y	s_1
i+1	x	β	y	s_2

$$\forall T \forall T_1 \forall C \forall C_1\ (\mathit{Symbol}[\alpha](T,C) \wedge \mathit{Cursor}(T,C_1) \wedge \prec(C,C_1) \wedge \mathit{Succ}(T,T_1) \rightarrow \mathit{Symbol}[\alpha](T_1,C))$$

$$\forall T \forall T_1 \forall C \forall C_1\ (\mathit{Symbol}[\alpha](T,C) \wedge \mathit{Cursor}(T,C_1) \wedge \prec(C_1,C) \wedge \mathit{Succ}(T,T_1) \rightarrow \mathit{Symbol}[\alpha](T_1,C))$$

Accepting Rule

Once we reach the accepting state we accept



$$\forall T\,(State[s_{acc}](T) \rightarrow Accept(T))$$

Defining *First*, *Succ* and $\prec$

- $First(0)$, $Succ(0,1)$, $Succ(1,2)$, $Succ(2,3)$, …, $Succ(N\text{-}2,N\text{-}1)$
- In fact, 0,…,N-1 are in binary form - assume the $N = 2^m$, where m = 3

  $First(0,0,0)$, $Succ(0,0,0,0,0,1)$, $Succ(0,0,1,0,1,0)$,…, $Succ(1,1,0,1,1,1)$

- Inductive definition of $First_i$ and $Succ_i$

  $D = \{First_1(0), Last_1(1), Succ_1(0,1)\}$

  $First_2(0,0)$, $Last_2(1,1)$, $Succ_2(0,0,0,1)$, $Succ_2(0,1,1,0)$, $Succ(1,0,1,1)$

$$\forall X\,(First_1(X) \rightarrow First_2(X,X))$$

$$\forall X\,(Last_1(X) \rightarrow Last_2(X,X))$$

Defining *First*, *Succ* and $\prec$

- *First*(0), *Succ*(0,1), *Succ*(1,2), *Succ*(2,3), …, *Succ*(*N*-2,*N*-1)
- In fact, 0,…,*N*-1 are in binary form - assume the $N = 2^m$, where m = 3

  First(0,0,0), *Succ*(0,0,0,0,0,1), *Succ*(0,0,1,0,1,0),…, *Succ*(1,1,0,1,1,1)

- Inductive definition of $First_i$ and $Succ_i$

$D = \{First_1(0), Last_1(1), Succ_1(0,1)\}$

$First_2(0,0)$, $Last_2(1,1)$, $Succ_2(0,0,0,1)$, $Succ_2(0,1,1,0)$, $Succ(1,0,1,1)$

$$\forall X \forall Y \forall Z\ (First_1(X), Succ_1(Y,Z) \rightarrow Succ_2(X,Y,X,Z))$$

$$\forall X \forall Y \forall Z\ (Last_1(X), Succ_1(Y,Z) \rightarrow Succ_2(X,Y,X,Z))$$

Defining *First*, *Succ* and $\prec$

- $First(0)$, $Succ(0,1)$, $Succ(1,2)$, $Succ(2,3)$, …, $Succ(N\text{-}2,N\text{-}1)$
- In fact, 0,…,N-1 are in binary form - assume $N = 2^m$, where m = 3

  $First(0,0,0)$, $Succ(0,0,0,0,0,1)$, $Succ(0,0,1,0,1,0)$,…, $Succ(1,1,0,1,1,1)$

- Inductive definition of $First_i$ and $Succ_i$

  $D = \{First_1(0), Last_1(1), Succ_1(0,1)\}$

  $First_2(0,0)$, $Last_2(1,1)$, $Succ_2(0,0,0,1)$, $Succ_2(0,1,1,0)$, $Succ(1,0,1,1)$

$$\forall X \forall Y \forall Z \forall W\ (Last_1(X), First_1(Y), Succ_1(Z,W) \rightarrow Succ_2(Z,X,W,Y))$$

Defining *First*, *Succ* and $\prec$

$D = \{First_1(0), Last_1(1), Succ_1(0,1)\}$

Inductive definition of $First_{i+1}$ and $Succ_{i+1}$:

$$\forall \mathbf{X} \forall \mathbf{Y}\ (Succ_i(\mathbf{X},\mathbf{Y}) \rightarrow Succ_{i+1}(Z,\mathbf{X},Z,\mathbf{Y}))$$

$$\forall \mathbf{X} \forall \mathbf{Y} \forall Z \forall W\ (Succ_1(Z,W) \wedge Last_i(\mathbf{X}) \wedge First_i(\mathbf{Y}) \rightarrow Succ_{i+1}(Z,\mathbf{X},W,\mathbf{Y}))$$

$$\forall \mathbf{X} \forall Z\ (First_1(Z) \wedge First_i(\mathbf{X}) \rightarrow First_{i+1}(Z,\mathbf{X}))$$

$$\forall \mathbf{X} \forall Z\ (Last_1(Z) \wedge Last_i(\mathbf{X}) \rightarrow Last_{i+1}(Z,\mathbf{X}))$$

Definition of $\prec_m$:

$$\forall \mathbf{X} \forall \mathbf{Y}\ (Succ_m(\mathbf{X},\mathbf{Y}) \rightarrow \prec_m(\mathbf{X},\mathbf{Y}))$$

$$\forall \mathbf{X} \forall \mathbf{Y} \forall \mathbf{Z}\ (Succ_m(\mathbf{X},\mathbf{Z}) \prec_m(\mathbf{Z},\mathbf{Y}) \rightarrow \prec_m(\mathbf{X},\mathbf{Y}))$$

Concluding EXPTIME-hardness of FULL

- Several rules but polynomially many $\Rightarrow$ feasible in polynomial time

- $D \land \Sigma \models \exists X\ Accept(X)$ iff M accepts I in at most N steps

- Can be formally shown by induction on the time steps

Corollary: BCQ-Answering under FULL is EXPTIME-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

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
  - $V = \{P \mid P \in \mathrm{sch}(\Sigma)\}$
  - $E = \{(P,R) \mid \forall X \forall Y\ (\ldots \wedge P(X,Y) \wedge \ldots \rightarrow \exists Z\ (\ldots \wedge R(X,Z) \wedge \ldots)) \in \Sigma\}$

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

- We denote ACYCLIC the class of acyclic existential rules

Acyclic Existential Rules

- Given Σ, we can decide in polynomial time whether $\Sigma \in$ ACYCLIC

- But, acyclicity is a global property - we have to consider Σ as a whole

  $\Rightarrow$ not closed under union

$$\forall X \forall Y\, (R(X,Y) \rightarrow P(Y))$$

$$\forall X\, (P(X) \rightarrow \exists Y\, R(X,Y))$$

each rule alone is acyclic, but

together form a cyclic set of rules

- Why the chase terminates?

Acyclic Existential Rules

- A stratification of Σ is a sequence of sets $\Sigma_1, \ldots, \Sigma_n$ such that, for some function $\mu: sch(\Sigma) \to \{1,...,n\}$:
  1. $\{\Sigma_1, \ldots, \Sigma_n\}$ is a partition of Σ
  2. For each predicate $P \in sch(\Sigma)$, all the rules with P in the head are in $\Sigma_{\mu(P)}$ (i.e., in the same set of the partition)
  3. If $\forall X \forall Y (\ldots \wedge P(X,Y) \wedge \ldots \to \exists Z (\ldots \wedge R(X,Z) \wedge \ldots)) \in \Sigma$, then $\mu(P) < \mu(R)$

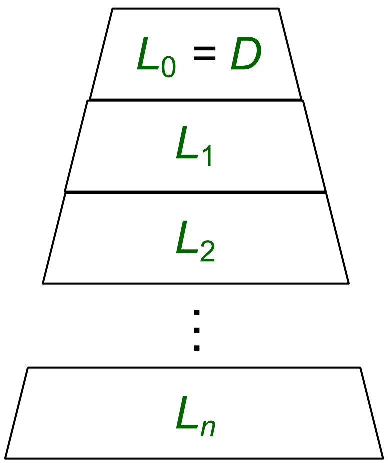
- Lemma: (1) Σ is stratifiable iff $\Sigma \in$ ACYCLIC

  (2) If there exists a path from P to R in PG(Σ), then $\mu(P) < \mu(R)$

- Thus, by exploiting the predicate graph, we can compute a stratification of Σ

Acyclic Existential Rules

- Consider $\Sigma \in$ **ACYCLIC**, and let $\Sigma_1, \ldots, \Sigma_n$ be a stratification of Σ
- Construct the chase by considering one stratum after the other starting from Σ_1

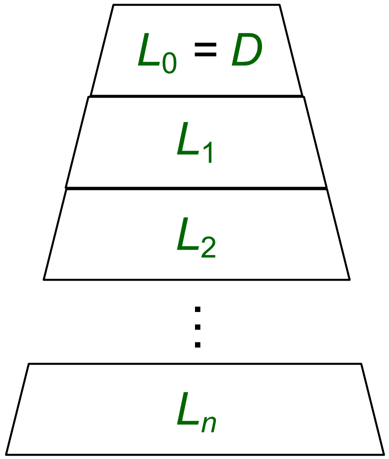


- For each $k \in \{1,\ldots,n\text{-}1\}$, $L_k = \text{chase}(L_{k-1},\Sigma_k)$
- n is finite $\Rightarrow$ the chase terminates

$\Rightarrow$ the naïve algorithm gives a decision procedure

…but, can we do better than the naïve algorithm?

The Naïve Algorithm for ACYCLIC



$|L_0| = 2$

$|L_1| = (|L_0|)^2$

		L_1
0	0	$\mathbf{z_{00}}$
0	1	$\mathbf{z_{01}}$
1	0	$\mathbf{z_{10}}$
1	1	$\mathbf{z_{11}}$

$D = \{P_0(0), P_0(1)\}$

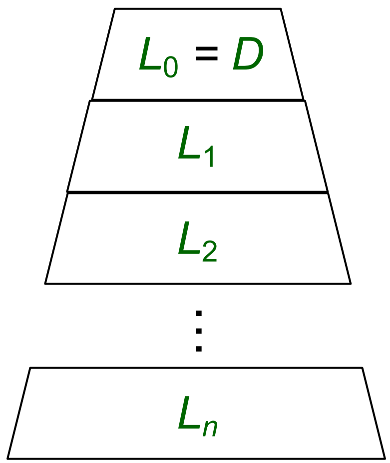
$\Sigma = \{\forall X \forall Y\ (P_0(X) \wedge P_0(Y) \rightarrow \exists Z\ (S_1(X,Y,Z) \wedge P_1(Z)))$

$\forall X \forall Y\ (P_1(X) \wedge P_1(Y) \rightarrow \exists Z\ (S_2(X,Y,Z) \wedge P_2(Z)))$

…

$\forall X \forall Y\ (P_{n\text{-}1}(X) \wedge P_{n\text{-}1}(Y) \rightarrow \exists Z\ (S_n(X,Y,Z) \wedge P_n(Z)))\}$

The Naïve Algorithm for ACYCLIC



$|L_0| = 2$

$|L_1| = (|L_0|)^2$

$|L_2| = (|L_1|)^2$

$D = \{P_0(0), P_0(1)\}$

$\Sigma = \{\forall X \forall Y\ (P_0(X) \land P_0(Y) \rightarrow \exists Z\ (S_1(X,Y,Z) \land P_1(Z)))$

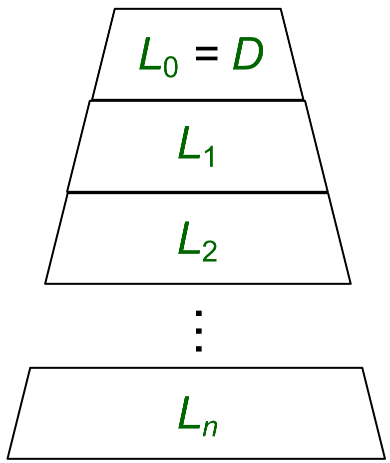
$\forall X \forall Y\ (P_1(X) \land P_1(Y) \rightarrow \exists Z\ (S_2(X,Y,Z) \land P_2(Z)))$

…

$\forall X \forall Y\ (P_{n\text{-}1}(X) \land P_{n\text{-}1}(Y) \rightarrow \exists Z\ (S_n(X,Y,Z) \land P_n(Z)))\}$

		L_2
z_{00}	z_{00}	$\mathbf{z_{0000}}$
z_{00}	z_{01}	$\mathbf{z_{0001}}$
z_{00}	z_{10}	$\mathbf{z_{0010}}$
z_{00}	z_{11}	$\mathbf{z_{0011}}$
z_{01}	z_{00}	$\mathbf{z_{0100}}$
z_{01}	z_{01}	$\mathbf{z_{0101}}$
z_{01}	z_{10}	$\mathbf{z_{0110}}$
z_{01}	z_{11}	$\mathbf{z_{0111}}$
z_{10}	z_{00}	$\mathbf{z_{1000}}$
z_{10}	z_{01}	$\mathbf{z_{1001}}$
z_{10}	z_{10}	$\mathbf{z_{1010}}$
z_{10}	z_{11}	$\mathbf{z_{1011}}$
z_{11}	z_{00}	$\mathbf{z_{1100}}$
z_{11}	z_{01}	$\mathbf{z_{1101}}$
z_{11}	z_{10}	$\mathbf{z_{1110}}$
z_{11}	z_{11}	$\mathbf{z_{1111}}$

The Naïve Algorithm for ACYCLIC



$|L_0| = 2$

$|L_1| = (|L_0|)^2$

$|L_2| = (|L_1|)^2$

$|L_n| = (|L_{n-1}|)^2$

		L_n
$z_{0\ldots0}$	$z_{0\ldots0}$	$\mathbf{z}_{0\ldots00\ldots0}$
…	…	…
$z_{1\ldots1}$	$z_{1\ldots1}$	$\mathbf{z}_{1\ldots11\ldots1}$

$D = \{P_0(0), P_0(1)\}$

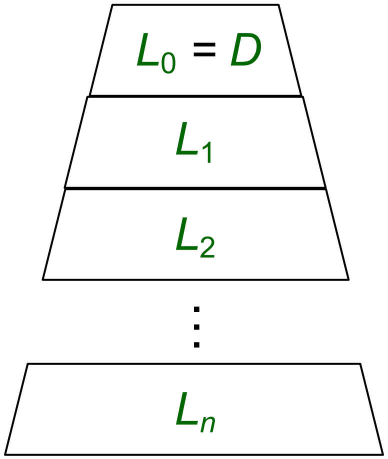
$\Sigma = \{\forall X \forall Y\ (P_0(X) \wedge P_0(Y) \rightarrow \exists Z\ (S_1(X,Y,Z) \wedge P_1(Z)))$

$\forall X \forall Y\ (P_1(X) \wedge P_1(Y) \rightarrow \exists Z\ (S_2(X,Y,Z) \wedge P_2(Z)))$

…

$\forall X \forall Y\ (P_{n-1}(X) \wedge P_{n-1}(Y) \rightarrow \exists Z\ (S_n(X,Y,Z) \wedge P_n(Z)))\}$

The Naïve Algorithm for ACYCLIC



$|L_0| = 2$

$|L_1| = (|L_0|)^2$

$|L_2| = (|L_1|)^2$

$|L_n| = (|L_{n-1}|)^2$

$|L_n| = 2^{(2\wedge n)}$

$D = \{P_0(0), P_0(1)\}$

$\Sigma = \{\forall X \forall Y\, (P_0(X) \wedge P_0(Y) \rightarrow \exists Z\, (S_1(X,Y,Z) \wedge P_1(Z)))$

$\forall X \forall Y\, (P_1(X) \wedge P_1(Y) \rightarrow \exists Z\, (S_2(X,Y,Z) \wedge P_2(Z)))$

…

$\forall X \forall Y\, (P_{n-1}(X) \wedge P_{n-1}(Y) \rightarrow \exists Z\, (S_n(X,Y,Z) \wedge P_n(Z)))\}$

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!!!

Data Complexity of ACYCLIC

Theorem: BCQ-Answering under ACYCLIC 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 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

Small Witness Property for ACYCLIC

Lemma: $\text{chase}(D,\Sigma) \models Q \quad \Rightarrow \quad$ there exists a chase sequence

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

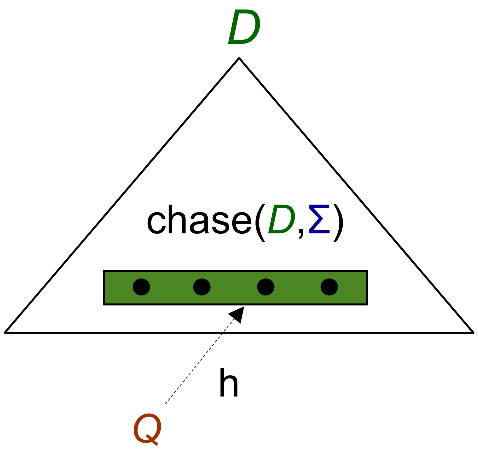
of D w.r.t. Σ with

$$n = \begin{cases} |Q| \cdot \lfloor(\text{maxbody}(\Sigma)^{|\text{sch}(\Sigma)|+1} - 1) / (\text{maxbody}(\Sigma) - 1)\rfloor, & \text{if maxbody}(\Sigma) > 1 \\ |Q| \cdot |\text{sch}(\Sigma)|, & \text{if maxbody}(\Sigma) = 1 \end{cases}$$

such that $J_n \models Q$

Proof:

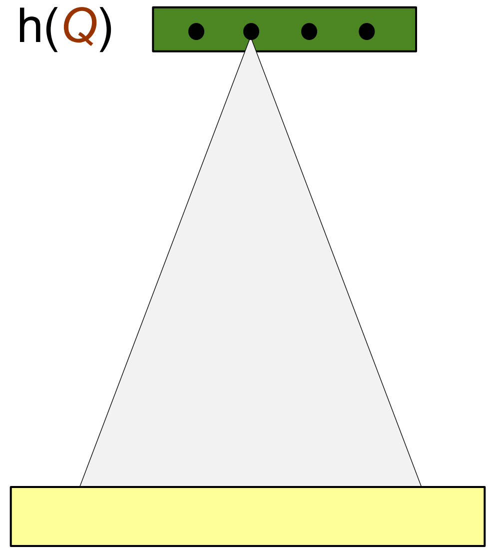
- By hypothesis, there exists a homomorphism h such that $h(Q) \subseteq \text{chase}(D, \Sigma)$



Small Witness Property for ACYCLIC

Proof (cont.):

- Let us focus on the image of the query



In the worst case, the shaded part forms a rooted tree:

1. With depth at most |sch(Σ)|
2. Each node has at most maxbody(Σ) children

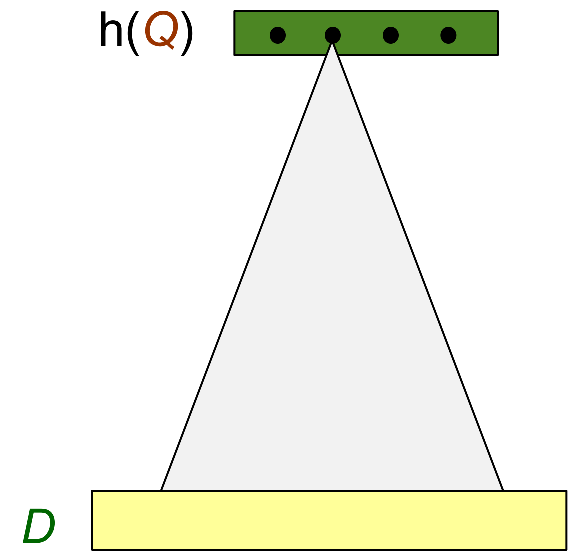
⇒ its size is at most

$$\begin{cases} \lfloor (\mathrm{maxbody}(\Sigma)^{|\mathrm{sch}(\Sigma)|+1} - 1) / (\mathrm{maxbody}(\Sigma) - 1) \rfloor, & \text{if } \mathrm{maxbody}(\Sigma) > 1 \\ |\mathrm{sch}(\Sigma)|, & \text{if } \mathrm{maxbody}(\Sigma) = 1 \end{cases}$$

Small Witness Property for ACYCLIC

Proof (cont.):

- Let us focus on the image of the query



Therefore, to entail the query we need at most

$$
\begin{cases}
|Q| \cdot \lfloor(\text{maxbody}(\Sigma)^{|\text{sch}(\Sigma)|+1} - 1) / (\text{maxbody}(\Sigma) - 1)\rfloor, & \text{if maxbody}(\Sigma) > 1 \\
|Q| \cdot |\text{sch}(\Sigma)|, & \text{if maxbody}(\Sigma) = 1
\end{cases}
$$

Combined Complexity of ACYCLIC

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

Proof: Consider a database D, a set $\Sigma \in$ ACYCLIC, 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

$$n = \begin{cases} |Q| \cdot \lfloor(\mathrm{maxbody}(\Sigma)^{|\mathrm{sch}(\Sigma)|+1} - 1) / (\mathrm{maxbody}(\Sigma) - 1)\rfloor, & \text{if } \mathrm{maxbody}(\Sigma) > 1 \\ |Q| \cdot |\mathrm{sch}(\Sigma)|, & \text{if } \mathrm{maxbody}(\Sigma) = 1 \end{cases}$$

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

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

Tiling Problem

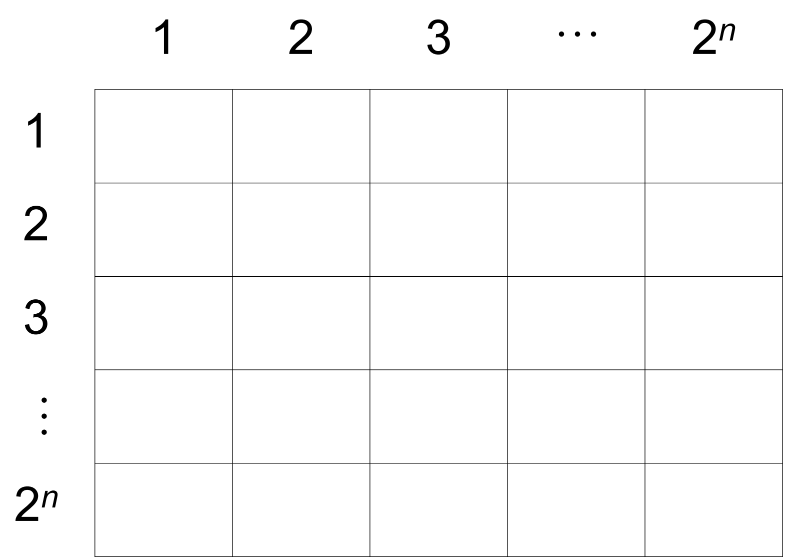
Tiling:

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,



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

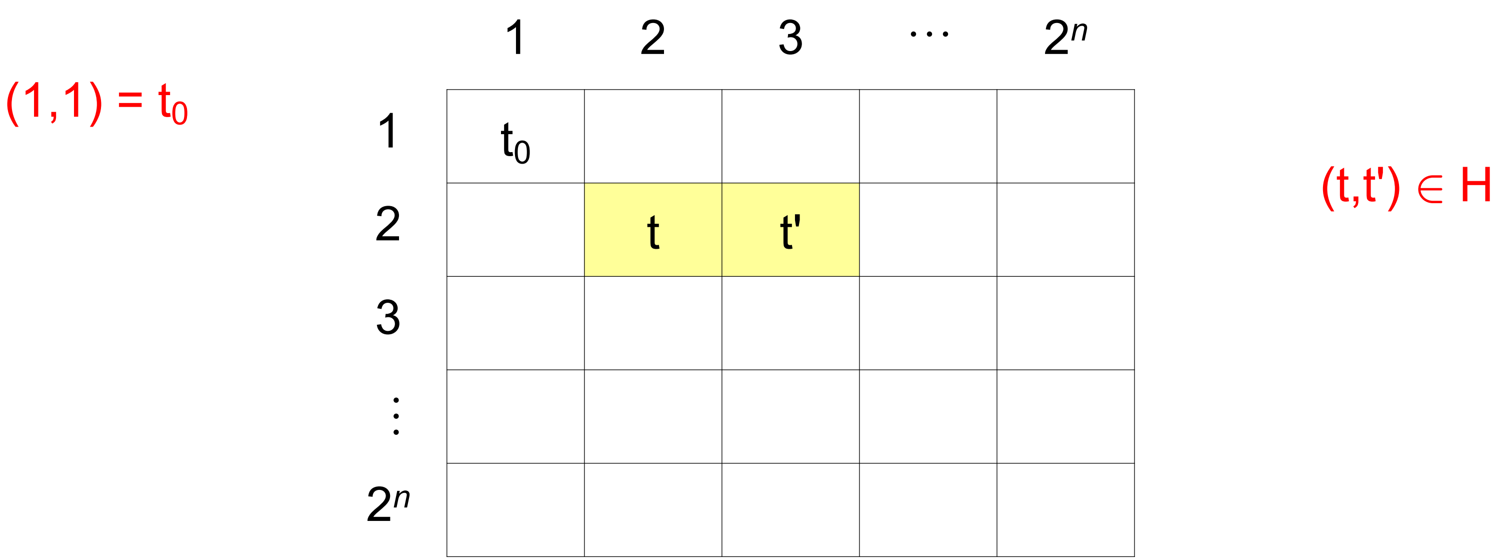
Tiling:

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,



Tiling Problem

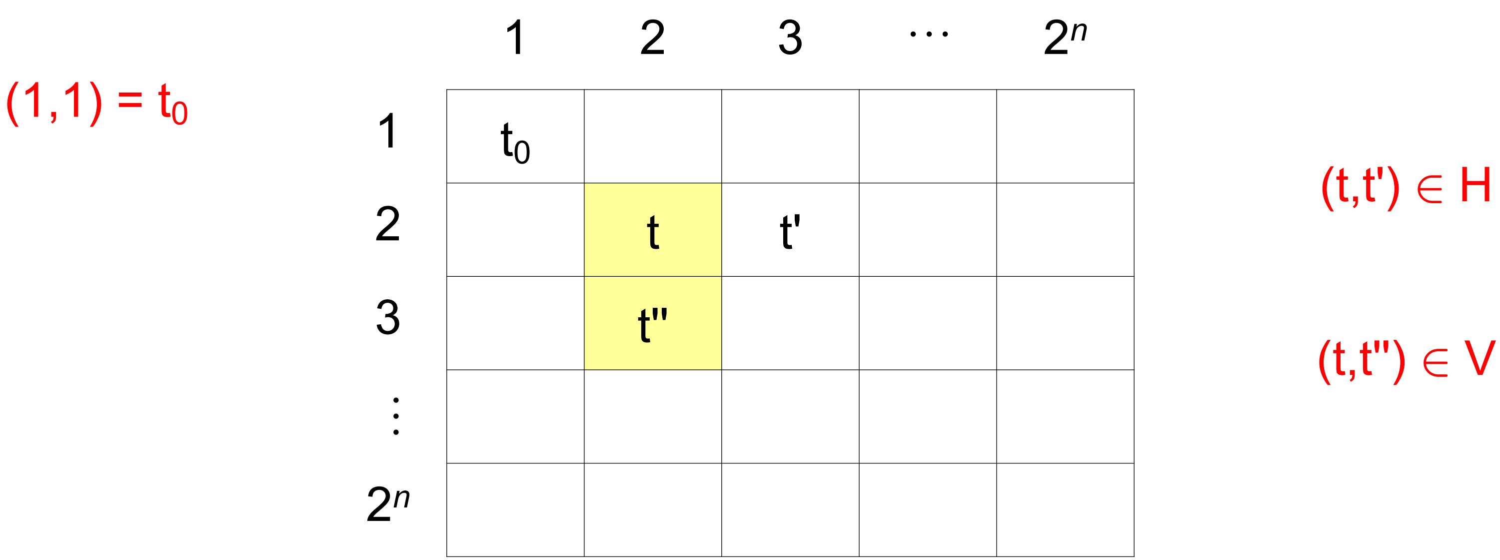
Tiling:

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,



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

$$\forall X_1 \forall X_2 \forall X_3 \forall X_4 \; (H(X_1,X_2) \land H(X_3,X_4) \land V(X_1,X_3) \land V(X_2,X_4) \rightarrow$$

$$\exists Y \; T_1(Y,X_1,X_1,X_2,X_3,X_4))$$

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

→

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

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

($\forall$-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
- $D \land \Sigma \models \exists X\ T(X,t_0)$ 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	Second part of our course

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