TECHNISCHE UNIVERSITÄT DRESDEN

International Center for Computational Logic

Computational Logic ∴ Group

Sebastian Rudolph
International Center for Computational Logic
TU Dresden

Existential Rules – Lecture 4

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

Syntax of Existential Rules

An existential rule is an expression

$$\forall \mathbf{X} \forall \mathbf{Y}\ (\underbrace{\varphi(\mathbf{X},\mathbf{Y})}_{\text{body}} \rightarrow \exists \mathbf{Z}\ \underbrace{\psi(\mathbf{X},\mathbf{Z})}_{\text{head}})$$

- $\mathbf{X}$,$\mathbf{Y}$ and $\mathbf{Z}$ are tuples of variables of **V**
- $\varphi(\mathbf{X},\mathbf{Y})$ and $\psi(\mathbf{X},\mathbf{Z})$ are (constant-free) conjunctions of atoms

…a.k.a. tuple-generating dependencies, and Datalog$^{\pm}$ rules

Syntax of Conjunctive Queries

A conjunctive query (CQ) is an expression

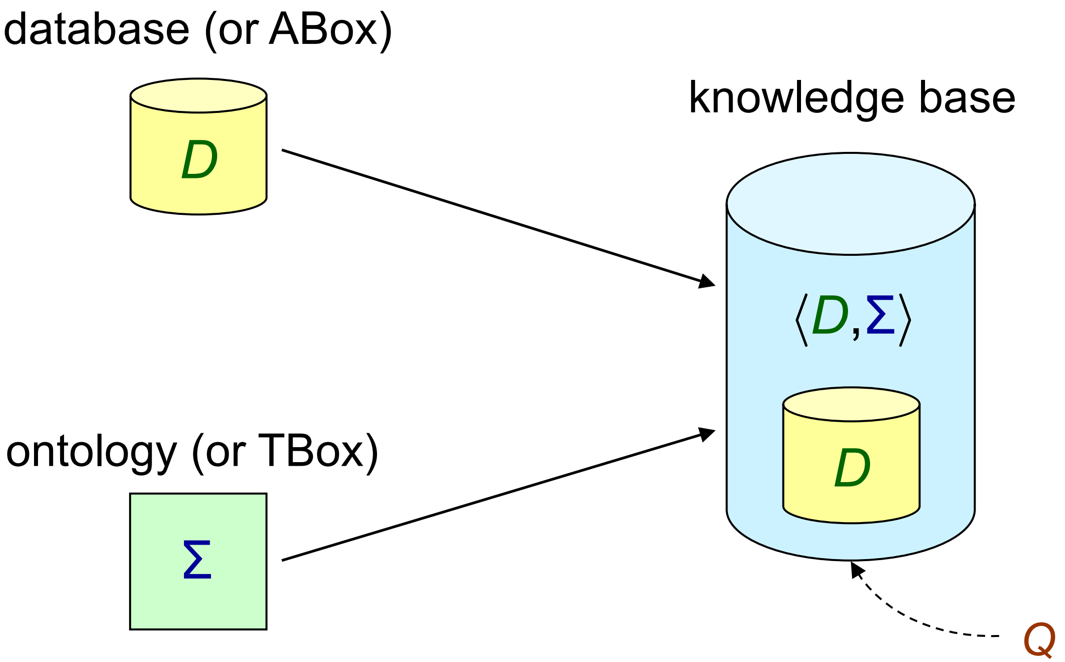
$$\exists \mathbf{Y} \, (\varphi(\mathbf{X},\mathbf{Y}))$$

- $\mathbf{X}$ and $\mathbf{Y}$ are tuples of variables of $\mathbf{V}$
- $\varphi(\mathbf{X},\mathbf{Y})$ is a conjunction of atoms (possibly with constants)

The most important query language used in practice

Forms the SELECT-FROM-WHERE fragment of SQL

Ontology-Based Query Answering (OBQA)



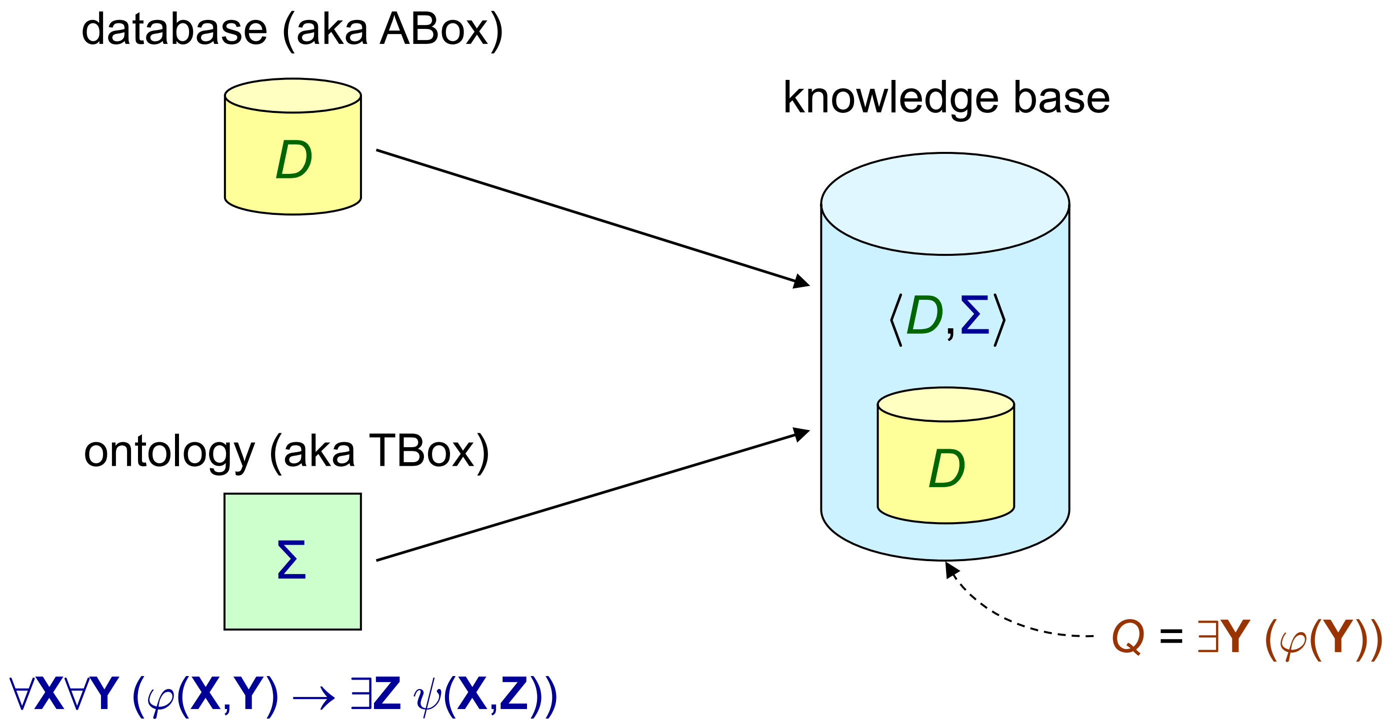
existential rules

$\forall \mathbf{X} \forall \mathbf{Y} \, (\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z} \, \psi(\mathbf{X},\mathbf{Z}))$

conjunctive queries

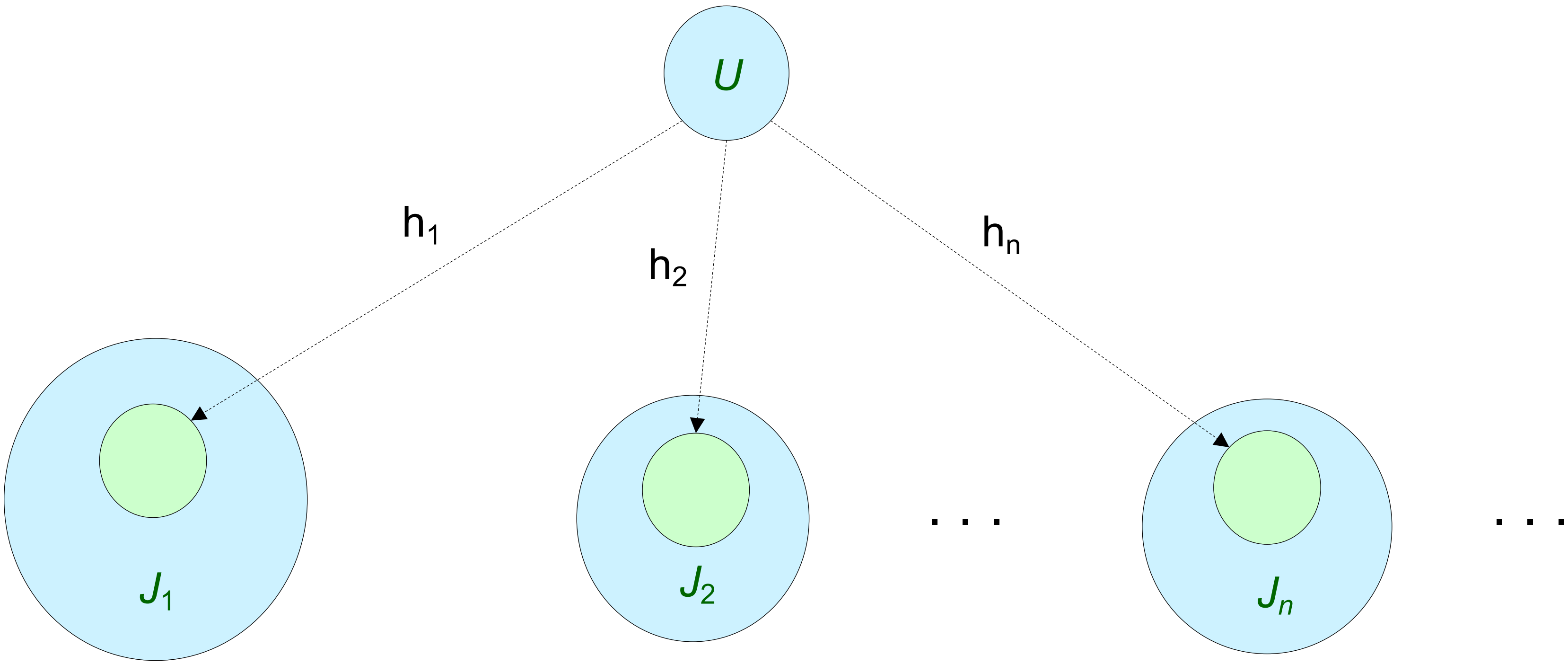
$\exists \mathbf{Y} \, (\varphi(\mathbf{X},\mathbf{Y}))$

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 $h_J(U) \subseteq J$

The Chase Procedure: Formal Definition

- Chase rule - the building block of the chase procedure

- A rule $\sigma = \forall \mathbf{X} \forall \mathbf{Y}\ (\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\ \psi(\mathbf{X},\mathbf{Z}))$ is applicable to instance J if:

    1. There exists a homomorphism h such that $h(\varphi(\mathbf{X},\mathbf{Y})) \subseteq J$
    2. There is no $g \supseteq h_{|\mathbf{X}}$ such that $g(\psi(\mathbf{X},\mathbf{Z})) \subseteq J$

- Let $J_{+} = J \cup \{g(\psi(\mathbf{X},\mathbf{Z}))\}$, where $g \supseteq h_{|\mathbf{X}}$ and $g(\mathbf{Z})$ are "fresh" nulls not in J

- The result of applying σ to J is J_{+}, denoted $J\langle\sigma,h\rangle J_{+}$ - single chase step

The Chase Procedure: Formal Definition

- A finite chase of D w.r.t. Σ is a finite sequence

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

where no rule from Σ is applicable in J_n .

Then, chase(D,Σ) is defined as the instance J_n

all applicable rules will eventually be applied

- An infinite chase of D w.r.t. Σ is a fair finite sequence

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

and chase(D,Σ) is defined as the instance $\cup_{k \geq 0} J_k$ (with $J_0 = D$)

least fixpoint of a monotonic operator - chase step

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) $\vDash Q$

- We can tame the first dimension of infinity by exploiting the chase procedure
- But, what about the second dimension of infinity? - the chase may be infinite

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

Deterministic Turing Machine (DTM)

$S \setminus \{s_{acc}\} \times \Lambda \to S \times \Lambda \times \{-1,0,+1\}$

accepting state

$$M \;=\; (S, \Lambda, \sqcup, \delta, s_0, s_{acc})$$

states

tape symbols

blank symbol

initial state

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

IF at some time instant τ the machine is in sate s_1, the cursor points to cell κ, and this cell contains α

THEN at instant τ+1 the machine is in state s_2, cell κ contains β, and the cursor points to cell κ+1

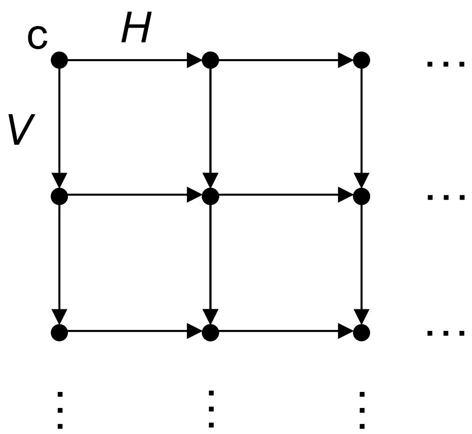
Undecidability of BCQ-Answering

Our Goal: Encode the computation of a DTM M with an empty tape

using a database D, a set Σ of existential rules, and a BCQ Q such that

$$D \wedge \Sigma \models Q \text{ iff } M \text{ accepts}$$

Build an Infinite Grid



k-th horizontal line represents the

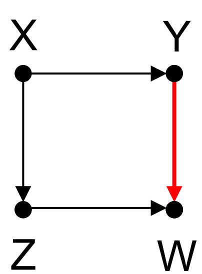
k-th configuration of the machine

$\forall X\ (\mathit{Start}(X) \rightarrow \mathit{Node}(X) \wedge \mathit{Initial}(X))$

$D = \{\mathit{Start}(c)\}$

fixes the origin of the grid

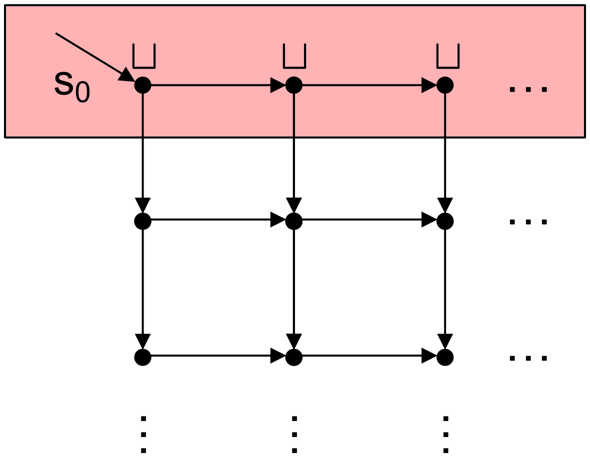
$\forall X\ (\mathit{Node}(X) \rightarrow \exists Y\ (H(X,Y) \wedge \mathit{Node}(Y)))$



$\forall X\ (\mathit{Node}(X) \rightarrow \exists Y\ (V(X,Y) \wedge \mathit{Node}(Y)))$

$\forall X \forall Y \forall Z \forall W\ (H(X,Y)\ H(Z,W)\ V(X,Z) \rightarrow V(Y,W))$

Initialization Rules

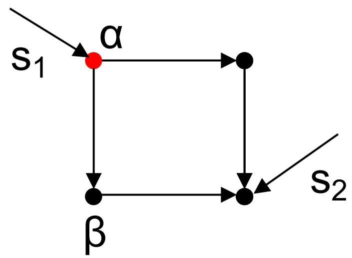


$\forall X \forall Y\ (\mathit{Initial}(X) \wedge H(X,Y) \rightarrow \mathit{Initial}(Y))$

$\forall X\ (\mathit{Start}(X) \rightarrow \mathit{Cursor}[s_0](X))$

$\forall X\ (\mathit{Initial}(X) \rightarrow \mathit{Symbol}[\sqcup](X))$

Transition Rules

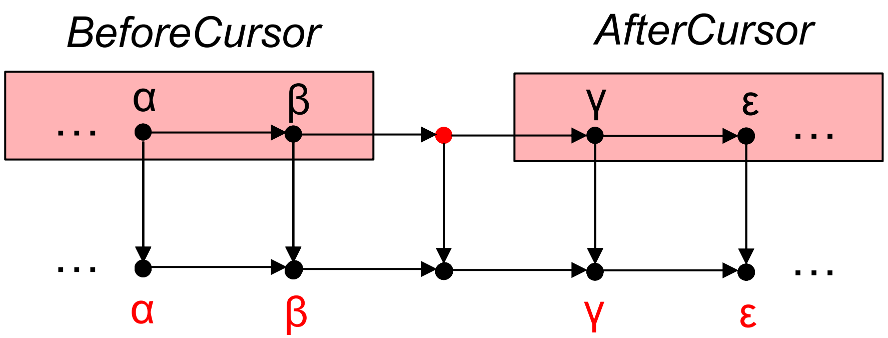


$\delta(s_1,\alpha) \;=\; (s_2,\beta,+1)$

$$\forall X \forall Y \forall Z\ (\mathit{Cursor}[s_1](X) \wedge \mathit{Symbol}[\alpha](X) \wedge V(X,Y) \wedge H(Y,Z) \rightarrow$$

$$\mathit{Cursor}[s_2](Z) \wedge \mathit{Symbol}[\beta](Y) \wedge \mathit{Mark}(X))$$

Inertia Rules



$\forall X \forall Y\ (\mathit{Mark}(X) \wedge H(X,Y) \rightarrow \mathit{AfterCursor}(Y))$

$\forall X \forall Y\ (\mathit{AfterCursor}(X) \wedge H(X,Y) \rightarrow \mathit{AfterCursor}(Y))$

$\forall X \forall Y\ (\mathit{AfterCursor}(X) \wedge \mathit{Symbol}[\alpha](X) \wedge V(X,Y) \rightarrow \mathit{Symbol}[\alpha](Y))$

…we have similar rules for the cells before the cursor

Accepting Rule

Once we reach the accepting state we accept

$$\forall X\ (\mathit{Cursor}[s_{acc}](X) \rightarrow \mathit{Accept}(X))$$

$D \wedge \Sigma \models \exists X\ \mathit{Accept}(X)$ iff the DTM *M* accepts

Undecidability of BCQ-Answering

Theorem: BCQ-Answering is undecidable

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

…syntactic restrictions are needed!!!

Gaining Decidability

By restricting the database

- $\{Start(c)\} \wedge \Sigma \vDash \quad Q$ iff the DTM M accepts
- The problem is undecidable already for singleton databases
- No much to do in this direction

By restricting the query language

- $D \wedge \Sigma \vDash \quad \exists X\, Accept(X)$ iff the DTM M accepts
- The problem is undecidable already for atomic queries
- No much to do in this direction

By restricting the ontology language

- Achieve a good trade-off between expressive power and complexity
- Field of intense research
- Any ideas?

… force the chase to terminate

What is the Source of Non-termination?

D

person(Alice)

Σ

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

$$chase(D,\Sigma) = D \cup \{hasParent(\text{Alice}, z_1), Person(z_1),$$
$$hasParent(z_1, z_2), Person(z_2),$$
$$hasParent(z_2, z_3), Person(z_3), \ldots$$

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

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

- 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?

Full Existential Rules

- Consider a database D and a set $\Sigma \in \mathsf{FULL}$

- $\mathrm{chase}(D,\Sigma) \subseteq \{P(c_1,\ldots,c_n) \mid \langle c_1,\ldots,c_n\rangle \in \mathrm{adom}(D)^n \;\text{ and }\; P \in \mathrm{sch}(\Sigma)\}$

active domain - constants occurring in D

schema - predicates occurring in Σ

- $|\mathrm{chase}(D,\Sigma)| \leq \underbrace{|\mathrm{sch}(\Sigma)| \cdot \overbrace{(|\mathrm{adom}(D)|)^{\mathrm{maxarity}}}^{\text{maximum number of tuples with terms of adom}(D)}}_{\text{maximum number of atoms with predicates of sch}(\Sigma)\text{ and terms of adom}(D)}$ maxarity $= \max_{P \in \mathrm{sch}(\Sigma)} \{\mathrm{arity}(P)\}$

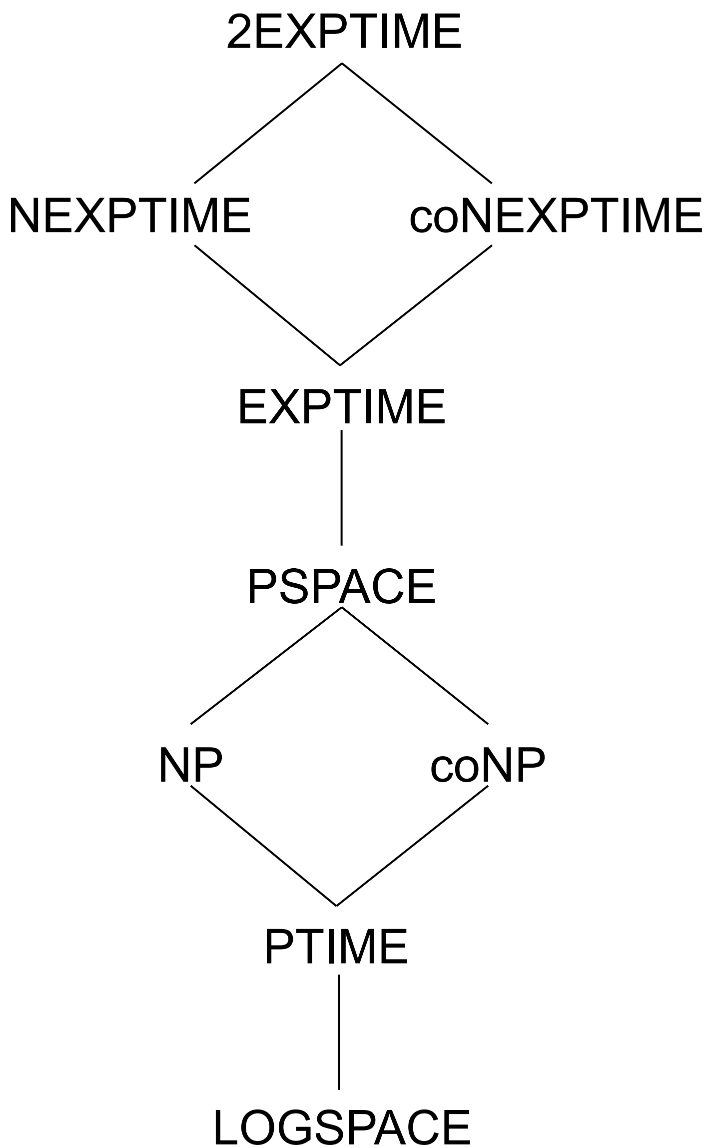
Complexity Measures for Query Answering

- 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

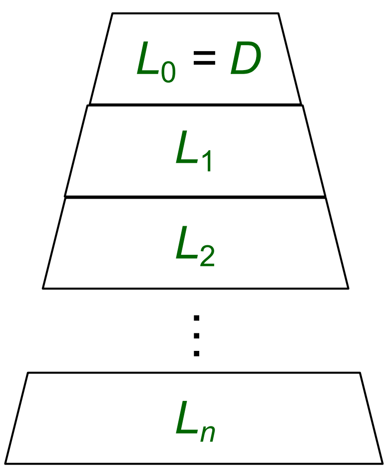
Theorem: BCQ-Answering under FULL is in PTIME w.r.t. the data complexity

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)

Step 1: We construct the chase level-by-level



- From L_k to L_{k+1}: for each $\sigma \in \Sigma$, find all the homomorphisms h such that h(body(σ)) $\subseteq L_k$, and add to L_k the set of atoms h(head(σ))
- Stop when $L_k = L_{k+1}$

$$|\Sigma| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxvariables}(\Sigma)} \cdot \mathrm{maxbody}(\Sigma) \cdot |L_k|$$

Data Complexity of FULL

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

Step 1: We construct the chase level-by-level in time

$$(k\text{-}1) \cdot |\Sigma| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxvariables}(\Sigma)} \cdot \mathrm{maxbody}(\Sigma) \cdot |L|$$

$$\text{where } \; k, |L| \;\leq\; |\mathrm{chase}(D,\Sigma)| \;\leq\; |\mathrm{sch}(\Sigma)| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxarity}}$$

Data Complexity of FULL

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

Step 2: By applying similar analysis, we can show that the existence of h can be checked in time

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

$$\text{where } |\mathrm{chase}(D,\Sigma)| \leq |\mathrm{sch}(\Sigma)| \cdot (|\mathrm{adom}(D)|)^{\mathrm{maxarity}}$$

Data Complexity of FULL

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

Consequently, in the worst case, the naïve algorithm runs in time

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

$$+$$

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

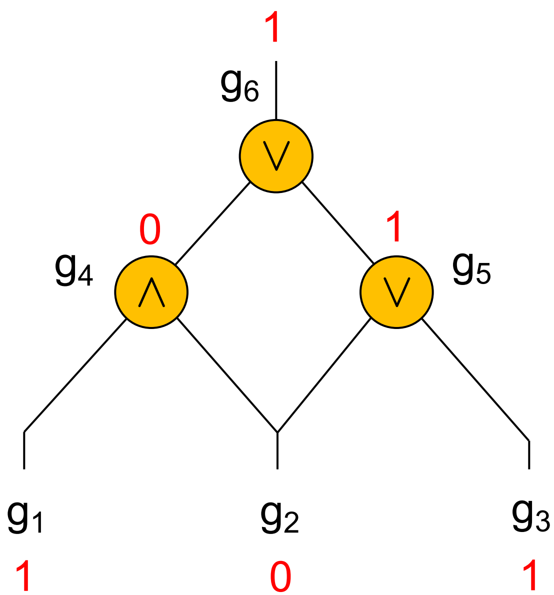
Data Complexity of FULL

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*

$T(g_1) \quad T(g_3)$

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

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

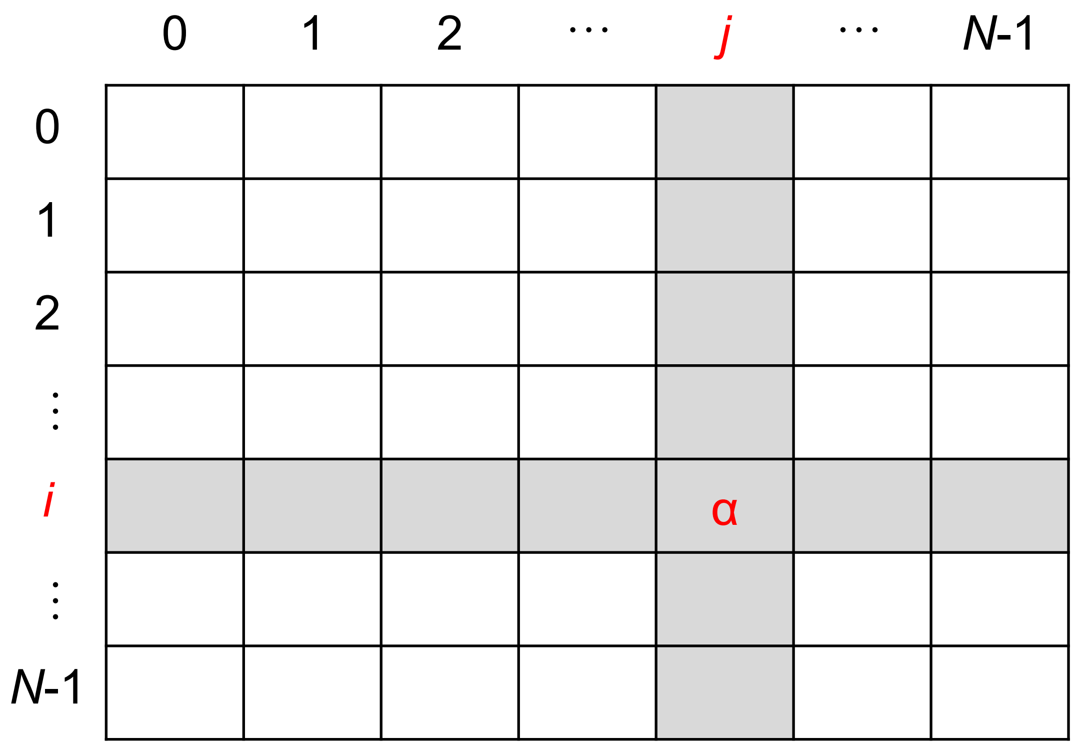
Proof : By simulating a deterministic exponential time Turing machine

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

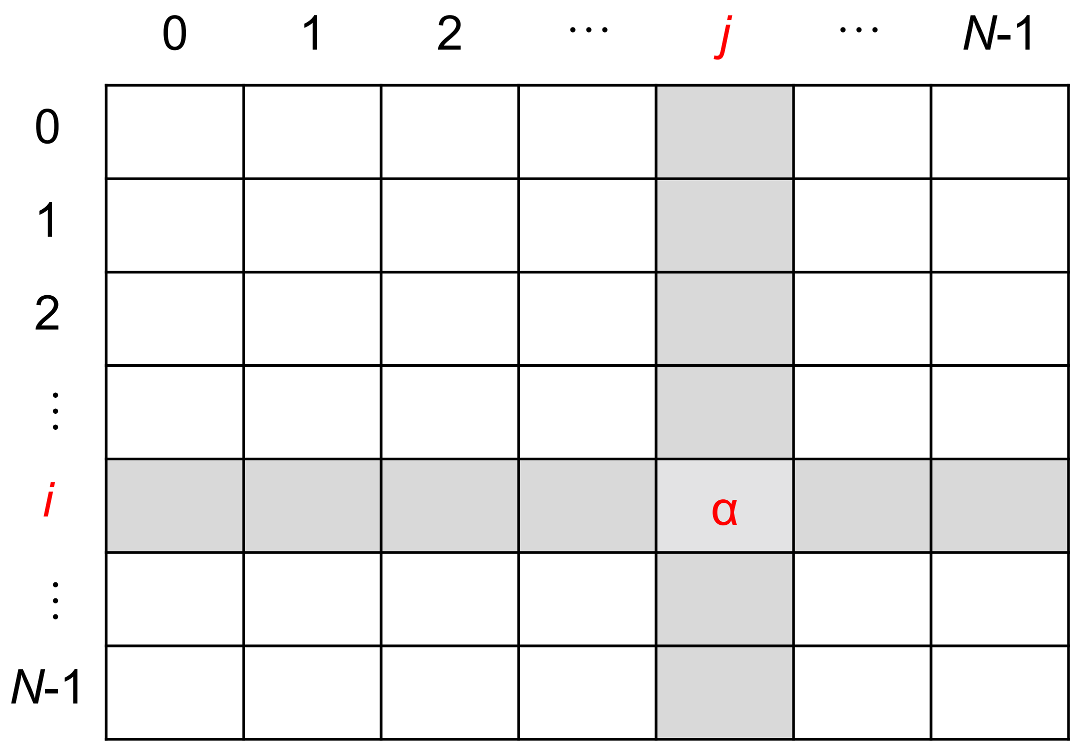
$D \wedge \Sigma \vDash Q$ iff M accepts I in at most $N = 2^m$ steps, where $m = |I|^k$

The Schema



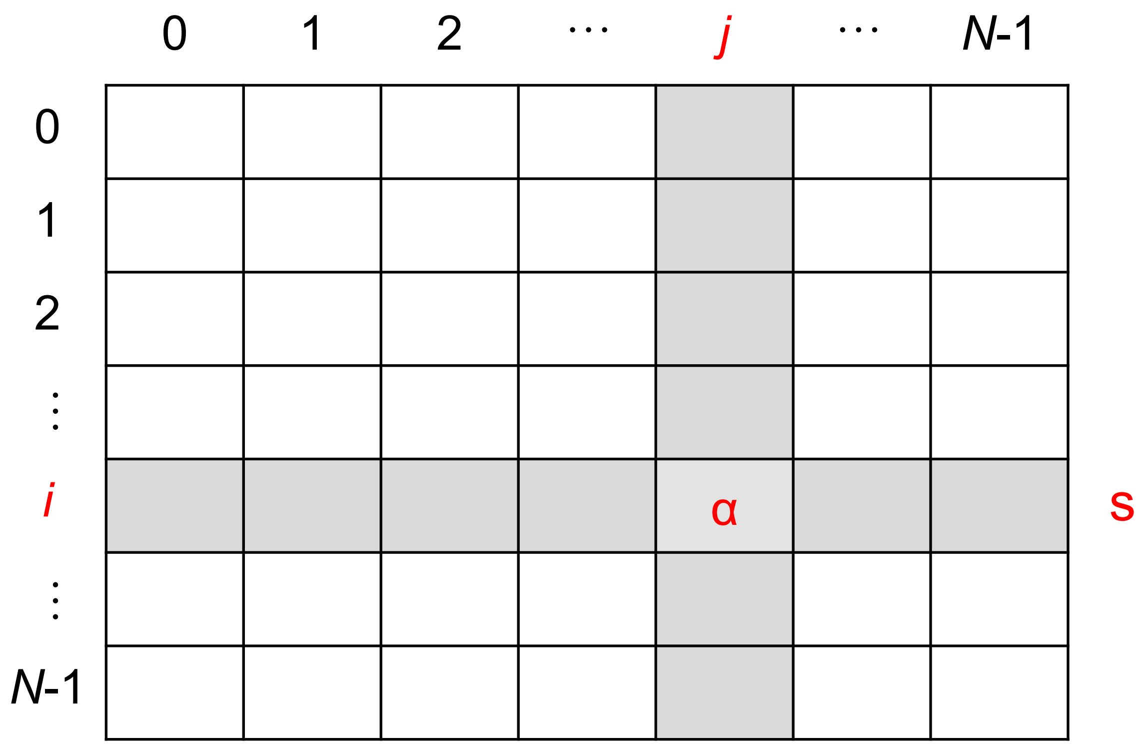
Symbol[α](*i*,*j*) - at time instant *i*, cell *j* contains α

The Schema



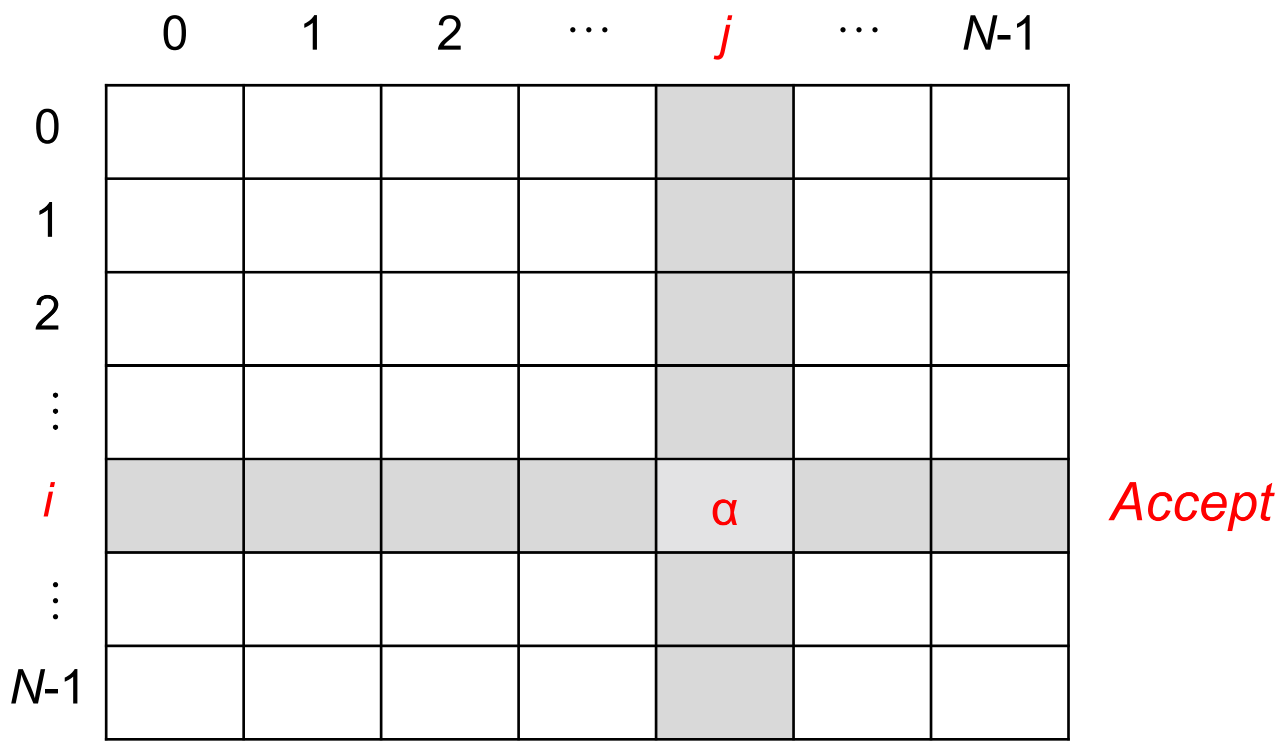
Cursor(*i*,*j*) - at time instant *i*, cursor points to cell *j*

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

	0	1	2	$\cdots$	j	$\cdots$	N-1
0							
1							
2							
$\vdots$							
i							
$\vdots$							
N-1							

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

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

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

Transition Rules

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

	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

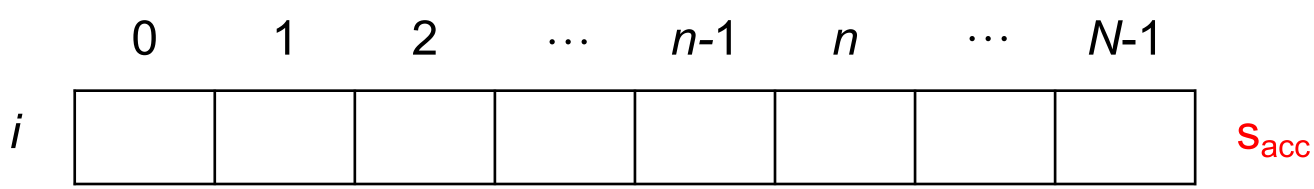
	j	$j+1$	$j+2$	
i	x	α	y	s_1
$i+1$	x	β	y	s_2

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

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

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, $0,\ldots,N\text{-}1$ 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 $Succ_i$

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

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

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

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

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*-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 $Succ_i$

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

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

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

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

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

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

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

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

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

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

Definition of $\prec_m$:

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

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

Concluding EXPTIME-hardness of FULL

- Several rules but polynomially many $\Rightarrow$ feasible in polynomial time
- $D \wedge \Sigma \models \exists X\ \mathit{Accept}(X)$ 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