Datalog and its Extensions for Semantic Web Databases

Georg Gottlob[1] Giorgio Orsi[1] Andreas Pieris[1] Mantas Šimkus[2]

[1]Department of Computer Science, University of Oxford, UK

[2]Institute of Information Systems, Vienna University of Technology, Austria

Reasoning Web Summer School, Vienna, Austria, September 3 - 8, 2012

Structure of the Lecture

- Relational Databases and Datalog
- Complexity of Datalog
- Datalog and Ontological Reasoning
- Datalog$^{\pm}$

Relational Databases

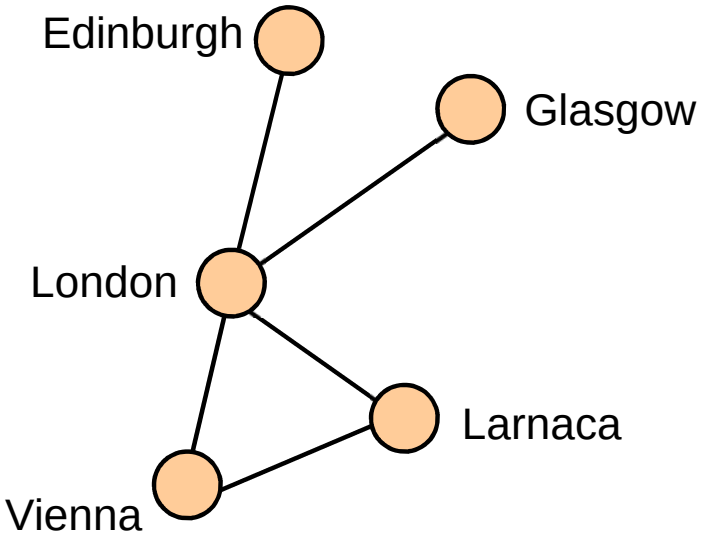
Predominant technology
for data storage and processing

Relational Databases

Predominant technology
for data storage and processing

"On the fly" example:

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS



Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Relational Databases (Terminology)

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Constants
(from a domain)

VIE, LHR, …

BA, U2, OS

Vienna, London, …

Relational Databases (Terminology)

Relations

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

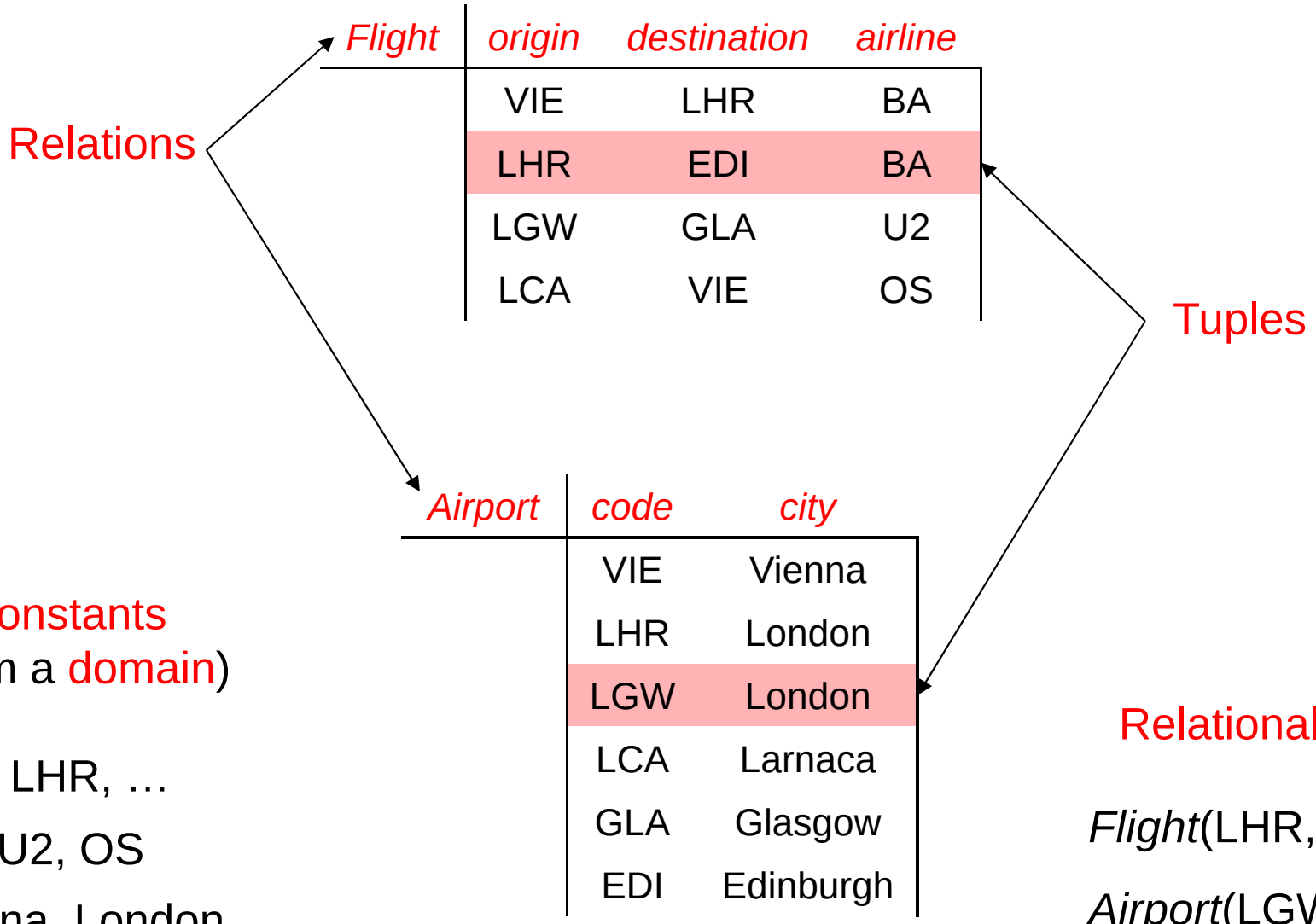
Constants
(from a domain)

VIE, LHR, …

BA, U2, OS

Vienna, London, …

Relational Databases (Terminology)



Constants
(from a domain)

VIE, LHR, …

BA, U2, OS

Vienna, London, …

Relational atoms

Flight(LHR,EDI,BA)

Airport(LGW,London)

Querying: Relational Algebra

List all the airlines

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

{BA, U2, OS}

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$\Pi_{airline}$ **Flight**

Querying: Relational Algebra

List the codes of the airports in London

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{LHR, LGW}

$$\Pi_{\texttt{code}}\ (\sigma_{\texttt{city = ``London''}}\ \texttt{Airport})$$

Querying: Relational Algebra

Which airlines fly directly from London to Glasgow?

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$$T_1 \leftarrow \sigma_{city = \text{“London”}} \ \textbf{Airport}$$

$$T_2 \leftarrow \sigma_{city = \text{“Glasgow”}} \ \textbf{Airport}$$

Querying: Relational Algebra

Which airlines fly directly from London to Glasgow?

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

T_1	*code*	*city*
	LHR	London
	LGW	London

T_2	*code*	*city*
	GLA	Glasgow

$$T_1 \leftarrow \sigma_{city = \text{“London”}} \ \textbf{Airport}$$

$$T_2 \leftarrow \sigma_{city = \text{“Glasgow”}} \ \textbf{Airport}$$

Querying: Relational Algebra

Which airlines fly directly from London to Glasgow?

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

T_1	*code*	*city*
	LHR	London
	LGW	London

T_2	*code*	*city*
	GLA	Glasgow

$$T_3 \leftarrow \texttt{Flight} \bowtie_{\texttt{origin = code}} T_1$$

Querying: Relational Algebra

Which airlines fly directly from London to Glasgow?

T_3	*origin*	*destination*	*airline*	*code*	*city*
	LHR	EDI	BA	LHR	London
	LGW	GLA	U2	LGW	London

T_2	*code*	*city*
	GLA	Glasgow

$$T_3 \leftarrow \texttt{Flight} \bowtie_{\texttt{origin = code}} T_1$$

Querying: Relational Algebra

Which airlines fly directly from London to Glasgow?

T_3	*origin*	*destination*	*airline*	*code*	*city*
	LHR	EDI	BA	LHR	London
	LGW	GLA	U2	LGW	London

T_2	*code*	*city*
	GLA	Glasgow

$$T_4 \leftarrow T_3 \bowtie_{\texttt{destination = code}} T_2$$

Querying: Relational Algebra

Which airlines fly directly from London to Glasgow?

T_4	*origin*	*destination*	*airline*	*code*	*city*	*code*	*city*
	LGW	GLA	U2	LGW	London	GLA	Glasgow

$$T_4 \leftarrow T_3 \bowtie_{\texttt{destination = code}} T_2$$

Querying: Relational Algebra

Which airlines fly directly from London to Glasgow?

T_4	*origin*	*destination*	*airline*	*code*	*city*	*code*	*city*
	LGW	GLA	U2	LGW	London	GLA	Glasgow

$$\Pi_{\texttt{airline}}\ T_4$$

Querying: FOL and SQL Representation

List all the airlines

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

FOL Representation

$\exists X \exists Y\ Flight(X,Y,Z)$

Z is free

SQL Representation

```
SELECT airline
FROM   Flight
```

Querying: FOL and SQL Representation

List the codes of the airports in London

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

FOL Representation

Airport(*X*,London)

SQL Representation

```
SELECT code
FROM   Airport
WHERE  city = “London”
```

Querying: FOL and SQL Representation

Which airlines fly directly from London to Glasgow?

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

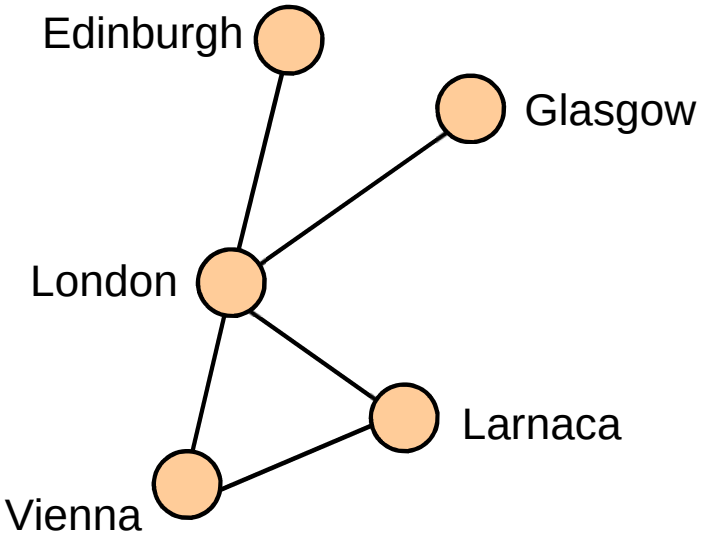
$$\exists X \exists Y \quad Airport(X,\text{London}) \wedge Airport(Y,\text{Glasgow}) \wedge Flight(X,Y,Z)$$

Z is free

```
SELECT F.airline
FROM   Flight as F,
       Airport as A1,
       Airport as A2
WHERE  A1.code = F.origin AND
       A2.code = F.destination AND
       A1.city = “London” AND
       A2.city = “Glasgow”
```

What we Cannot Ask?

Is Glasgow reachable from Vienna?



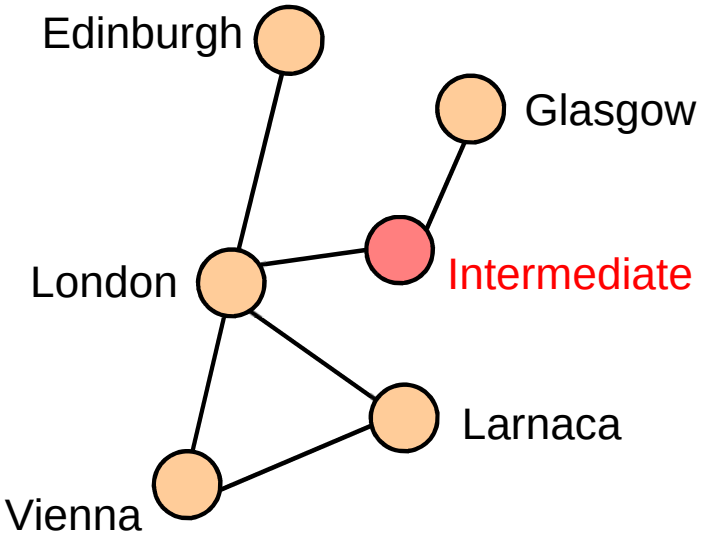
Looks like this FOL query can do the job…

$\exists X \exists Y \exists Z \exists W \exists V \; \mathit{Airport}(X,\text{Vienna}) \wedge \mathit{Airport}(Y,\text{Glasgow}) \wedge \mathit{Flight}(X,Z,W) \wedge \mathit{Flight}(Z,Y,V)$

YES

What we Cannot Ask?

Is Glasgow reachable from Vienna?



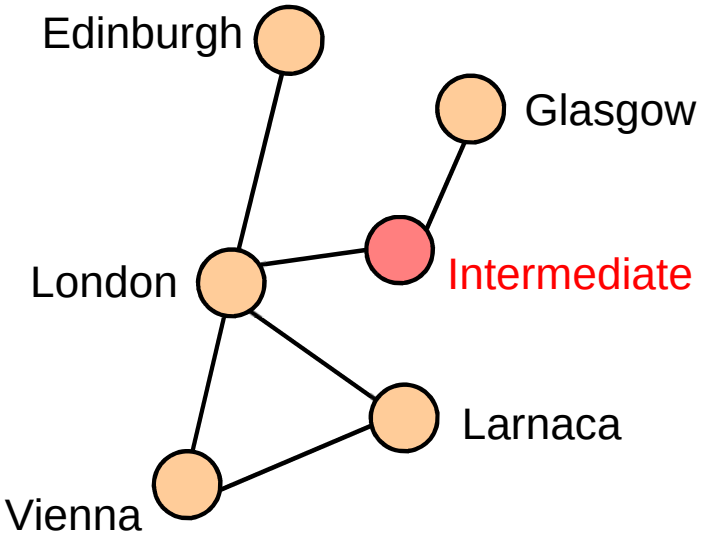
Looks like this FOL query can do the job…

$\exists X \exists Y \exists Z \exists W \exists V \; \mathit{Airport}(X,\text{Vienna}) \wedge \mathit{Airport}(Y,\text{Glasgow}) \wedge \mathit{Flight}(X,Z,W) \wedge \mathit{Flight}(Z,Y,V)$

NO

What we Cannot Ask?

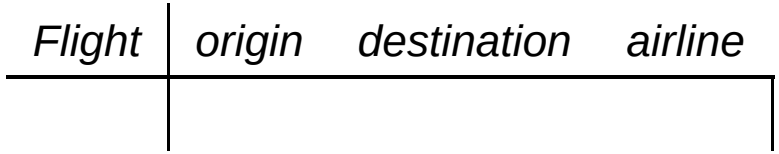
Is Glasgow reachable from Vienna?

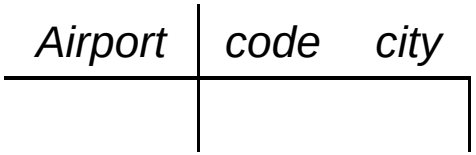


A possible strategy:

- List all the pairs of airports (A_1,A_2) such that A_2 is reachable from A_1
- Is their a pair (A_1,A_2) such that A_1 is in Vienna and A_2 is in Glasgow?

What we Cannot Ask?





– List all the pairs of airports (A_1,A_2) such that A_2 is reachable from A_1

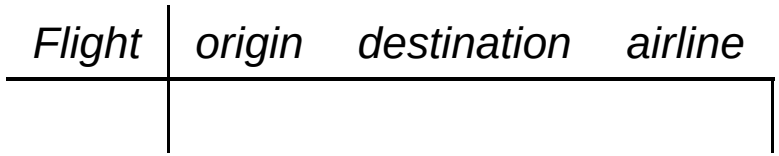
Reachable(*X*,*Y*) ← *Flight*(*X*,*Y*,*Z*)

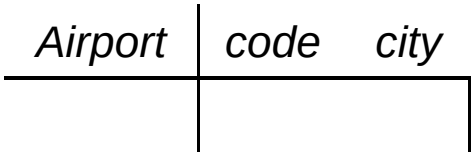
Reachable(*X*,*W*) ← *Flight*(*X*,*Y*,*Z*), *Reachable*(*Y*,*W*)

– Is their a pair (A_1,A_2) such that A_1 is in Vienna and A_2 is in Glasgow?

Ans() ← *Airport*(*X*,Vienna), *Airport*(*Y*,Glasgow), *Reachable*(*X*,*Y*)

What we Cannot Ask?





– List all the pairs of airports (A_1,A_2) such that A_2 is reachable from A_1

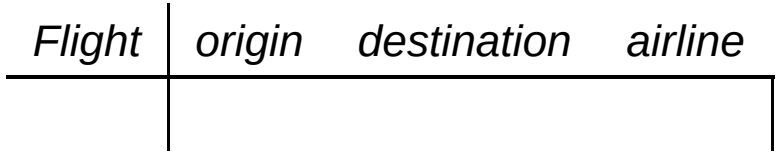
Reachable(*X*,*Y*) ← *Flight*(*X*,*Y*,*Z*)

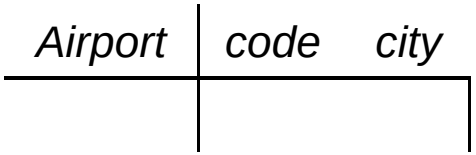
Reachable(*X*,*W*) ← *Flight*(*X*,*Y*,*Z*), *Reachable*(*Y*,*W*)

RECURSION

RA, FOL, SQL not enough

What we Cannot Ask?





– List all the pairs of airports (A_1,A_2) such that A_2 is reachable from A_1

Reachable(*X*,*Y*) ← *Flight*(*X*,*Y*,*Z*)

Reachable(*X*,*W*) ← *Flight*(*X*,*Y*,*Z*), *Reachable*(*Y*,*W*)

DATALOG

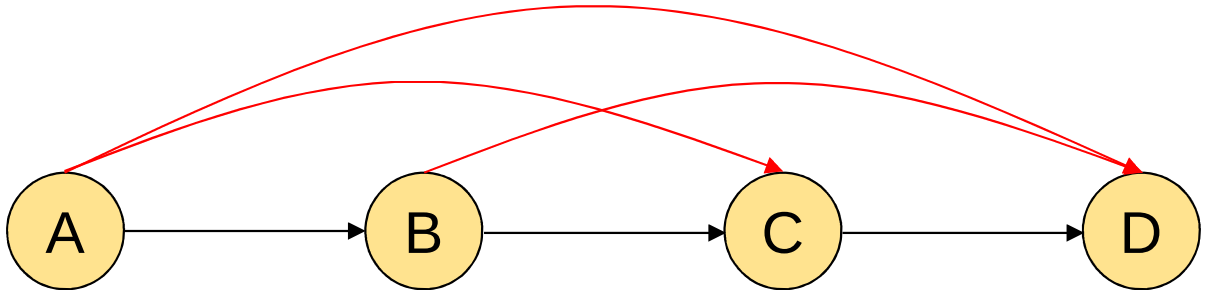
Select-Project-Join + Recursion

Datalog at First Glance

Transitive closure of a graph:

$$TrClosure(X,Y) \leftarrow Graph(X,Y)$$

$$TrClosure(X,Y) \leftarrow Graph(X,Z), TrClosure(Z,Y)$$



Datalog at First Glance

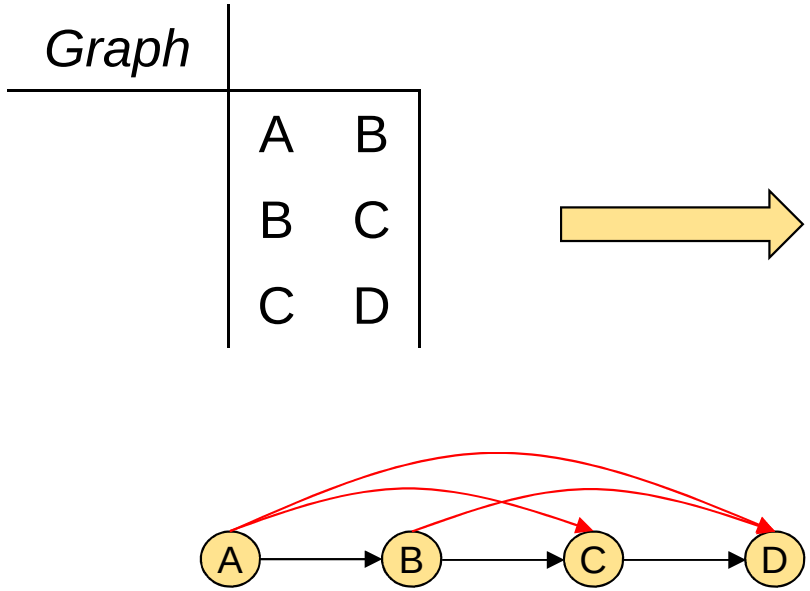
Transitive closure of a graph:

$$TrClosure(X,Y) \leftarrow Graph(X,Y)$$

$$TrClosure(X,Y) \leftarrow Graph(X,Z), TrClosure(Z,Y)$$

Graph

A	B
B	C
C	D

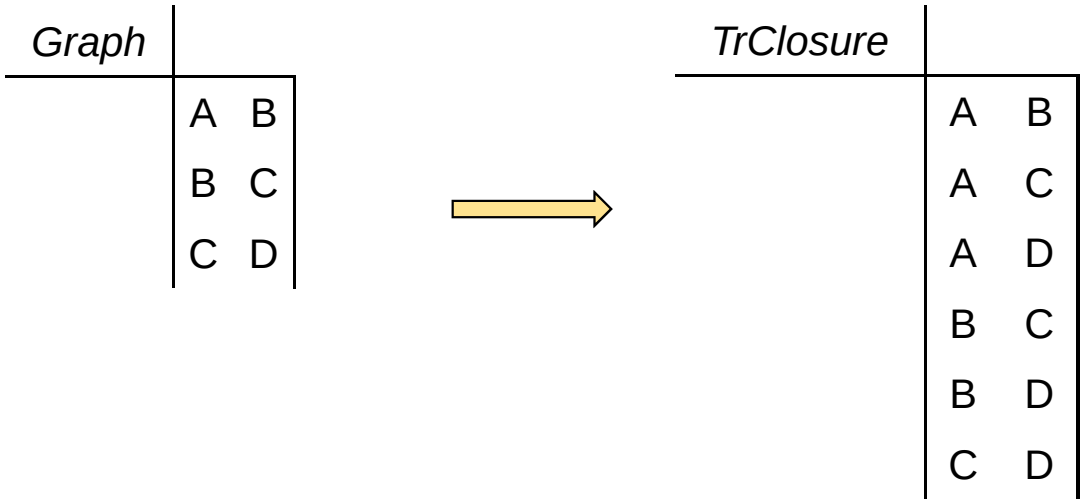


TrClosure

A	B
A	C
A	D
B	C
B	D
C	D

Datalog at First Glance

- Semantics - a mapping from instances over body-relations to instances over head-relations



- Equivalent approaches for defining the semantics:
  - Model-Theoretic - logical sentences asserting a property of the result
  - Fixpoint - solution of a fixpoint equation
  - Proof-Theoretic - obtaining proofs of facts

Datalog

– Formal Syntax

– Fixpoint Semantics

– Complexity Results

Note: For details on the model-theoretic and proof-theoretic semantics see:

- *Foundations of Databases* by Abiteboul, Hull and Vianu
- *Logic Programming and Databases* by Ceri, Gottlob and Tanca

Syntax of Datalog

A Datalog rule is an expression

$$\underbrace{R_0(\mathbf{X_0})}_{\text{head}} \leftarrow \underbrace{R_1(\mathbf{X_1}),\ldots,R_n(\mathbf{X_n})}_{\text{body}}$$

- $n \geq 0$ (empty body)
- $R_0,\ldots,R_n$ are relation symbols (or predicates)
- $\mathbf{X_0},\ldots,\mathbf{X_n}$ are tuples of terms (constants or variables)
- Safe - each variable occurring in head must occur in body

Syntax of Datalog

- Datalog program P - a (finite) set of Datalog rules
- Extensional predicate - does not occur in the head of a rule of P
- Intensional predicate - occurs in the head of some rule of P
- $edb(P)$ - extensional predicates of P
- $idb(P)$ - intensional predicates of P
- $sch(P)$ - the set of predicates $edb(P) \cup idb(P)$

Syntax of Datalog

- Datalog program *P* - a (finite) set of Datalog rules
- Extensional predicate - does not occur in the head of a rule of *P*
- Intensional predicate - occurs in the head of some rule of *P*
- *edb*(*P*) - extensional predicates of *P*
- *idb*(*P*) - intensional predicates of *P*

} not necessary

- *sch*(*P*) - the set of predicates *edb*(*P*) ∪ *idb*(*P*)

Example: Recursive Program *P*

Is Glasgow reachable from Vienna?

Flight	*origin*	*destination*	*airline*

Airport	*code*	*city*

$$\mathit{Reachable}(X,Y) \leftarrow \mathit{Flight}(X,Y,Z)$$

$$\mathit{Reachable}(X,W) \leftarrow \mathit{Flight}(X,Y,Z), \mathit{Reachable}(Y,W)$$

$$\mathit{Ans}() \leftarrow \mathit{Airport}(X,\text{Vienna}), \mathit{Airport}(Y,\text{Glasgow}), \mathit{Reachable}(X,Y)$$

$$\mathit{sch}(P) = \{\mathit{Flight}, \mathit{Airport}, \mathit{Reachable}, \mathit{Ans}\}$$

$$\mathit{edb}(P) = \{\mathit{Flight}, \mathit{Airport}\}$$

$$\mathit{idb}(P) = \{\mathit{Reachable}, \mathit{Ans}\}$$

Fixpoint Semantics

– Relies on the immediate consequence operator T_P

– Given a database D and a Datalog program P, an atom $R(c_1,\ldots,c_n)$ is an immediate consequence for D and P if:

- $R(c_1,\ldots,c_n) \in D$, or

- exists $R_0(\mathbf{X_0}) \leftarrow R_1(\mathbf{X_1}),\ldots,R_n(\mathbf{X_n})$ in P, and a homomorphism h:

$$\{h(R_1(\mathbf{X_1})),\ldots,h(R_n(\mathbf{X_n}))\} \subseteq D \quad \text{and} \quad h(R_0(\mathbf{X_0})) = R(c_1,\ldots,c_n)$$

– T_P - mapping from databases for $sch(P)$ to databases for $sch(P)$

– $T_P(D)$ = { immediate consequences for D and P }

Fixpoint Semantics

– A crucial fact - for each *P* and *D* for *edb*(*P*):

T_P has a minimum fixpoint containing *D*

I is a fixpoint of T_P if $T_P(I) = I$

Note: For the proof see, e.g., Theorem 12.3.2
in *Foundations of Databases*

– The semantics of *P* on *D*, denoted *P*(*D*), is this minimum fixpoint

– How do we compute it?

Fixpoint Semantics

- $T_{P,0}(I) = I$ and $T_{P,i+1}(I) = T_P(T_{P,i}(I))$
- Compute $T_{P,\omega}(I) = \cup_{i \geq 0} T_{P,i}(I)$

Fixpoint Semantics: Example

– Let P the program which computes the transitive closure of a graph:

$$TrClosure(X,Y) \leftarrow Graph(X,Y)$$

$$TrClosure(X,Y) \leftarrow Graph(X,Z), TrClosure(Z,Y)$$

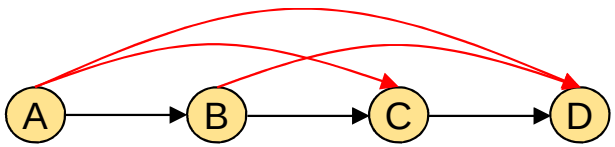
– Consider the input database $D = \{Graph(A,B), Graph(B,C), Graph(C,D)\}$

– $T_{P,1}(D) = D \cup \{TrClosure(A,B), TrClosure(B,C), TrClosure(C,D)\}$

$T_{P,2}(D) = T_{P,1}(D) \cup \{TrClosure(A,C), TrClosure(B,D)\}$

$T_{P,3}(D) = T_{P,2}(D) \cup \{TrClosure(A,D)\}$

$T_{P,4}(D) = T_{P,3}(D)$



– Thus, $T_{P,\omega}(D) = T_{P,3}(D)$

Fixpoint Semantics

- $T_{P,0}(I) = I$ and $T_{P,i+1}(I) = T_P(T_{P,i}(I))$
- Compute $T_{P,\omega}(I) = \cup_{i \geq 0} T_{P,i}(I)$
- $T_{P,\omega}(D)$ is the minimum fixpoint of T_P containing D
- $T_{P,\omega}(D)$ is the $\subseteq$-minimal model of P containing D

Note: For the proof see, e.g., Theorem 12.3.4 in *Foundations of Databases*

Model-Theoretic Approach

Transitive closure of a graph:

$$TrClosure(X,Y) \leftarrow Graph(X,Y)$$

$$TrClosure(X,Y) \leftarrow Graph(X,Z), TrClosure(Z,Y)$$

Graph	
A	B
B	C
C	D

TrClosure	
A	B
A	C
A	D
B	C
B	D
C	D

$\subseteq$-minimal model

M	
A	A
A	B
A	C
A	D
⋮	⋮
D	D

not $\subseteq$-minimal model

Complexity of Datalog

– Fact inference problem:

  - **Input**: program P, database D for $edb(P)$, atom α

  - **Question**: $P \cup D \models \alpha$, or, equivalently, $\alpha \in P(D)$?

– Data complexity - P fixed, D part of the input
  [Vardi, STOC 1982]

– Combined complexity - both P and D part of the input
  [Vardi, STOC 1982]

Data Complexity of Datalog

Theorem: Datalog is PTIME-complete in data complexity

Proof (in PTIME):

– consider a program P and a database D

– $|P(D)| \leq |sch(P)| \cdot \underbrace{(|dom(P) \cup dom(D)|)^{maxarity}}$

maximum number of tuples using constants of $dom(P) \cup dom(D)$

$dom(P)$ - constants in P

$dom(D)$ - constants in D

– *maxarity* is a constant $\Rightarrow$ $P(D)$ can be constructed in PTIME

Data Complexity of Datalog

Theorem: Datalog is PTIME-complete in data complexity

Proof (PTIME-hard): reduction from fact inference for propositional LP(2)

Data Complexity of Datalog

– Propositional LP(2) - set of rules $R_0 \leftarrow R_1, R_2$

– Fact inference for propositional LP(2):

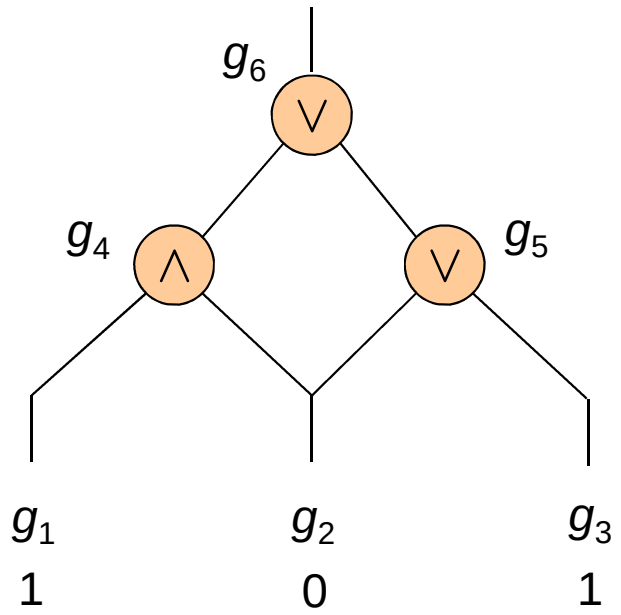
- **Input**: propositional LP(2) program P, propositional atom Q

- **Question**: $P \models Q$?

PTIME-hardness of LP(2)

Theorem: LP(2) is PTIME-hard

Proof: Logspace reduction from Monotone Circuit Value Problem

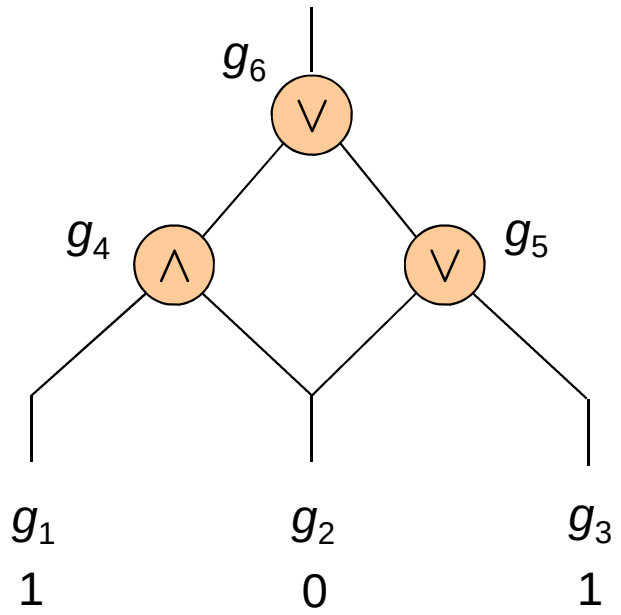


Does the circuit evaluate to *true*?

PTIME-hardness of LP(2)

Theorem: LP(2) is PTIME-hard

Proof: Logspace reduction from Monotone Circuit Value Problem



encoding of the circuit as LP(2) program P

$g_6 \leftarrow g_4$
$g_6 \leftarrow g_5$

$g_4 \leftarrow g_1, g_2$
$g_5 \leftarrow g_2$
$g_5 \leftarrow g_3$

$g_1 \leftarrow$
$g_3 \leftarrow$

Does the circuit evaluate to *true*?

Circuit evaluates to *true* iff $P \models g_6$

Data Complexity of Datalog

– Propositional LP(2) - set of rules $R_0 \leftarrow R_1, R_2$

– Fact inference for propositional LP(2):

- **Input**: propositional LP(2) program P, propositional atom Q

- **Question**: $P \models Q$?

– PTIME-hard - we can also simulate a PTIME Turing machine

see, e.g., [Dantsin, Eiter, Gottlob & Voronkov, ACM Computing Surveys 2001]

Data Complexity of Datalog

Theorem: Datalog is PTIME-complete in data complexity

Proof (PTIME-hard): reduction from fact inference for propositional LP(2)

– For each $R_0 \leftarrow$ add in D the atom $True(R_0)$
– For each $R_0 \leftarrow R_1, R_2$ add in D the atom $S(R_0, R_1, R_2)$

} encode program P in database D

– Construct the fixed Datalog program P_{DAT}:

$$T(X) \leftarrow True(X)$$
$$T(Z) \leftarrow T(X), T(Y), S(Z, X, Y)$$

} meta-interpreter for LP(2)

– $P \models Q$ iff $T(Q) \in P_{DAT}(D)$

Combined Complexity of Datalog

Theorem: Datalog is EXPTIME-complete in combined complexity

Proof (in EXPTIME):

- consider a program P and a database D
- $|P(D)| \leq |sch(P)| \cdot \underbrace{(|dom(P) \cup dom(D)|)^{maxarity}}$

  maximum number of tuples using constants of $dom(P) \cup dom(D)$
- $P(D)$ can be constructed in EXPTIME

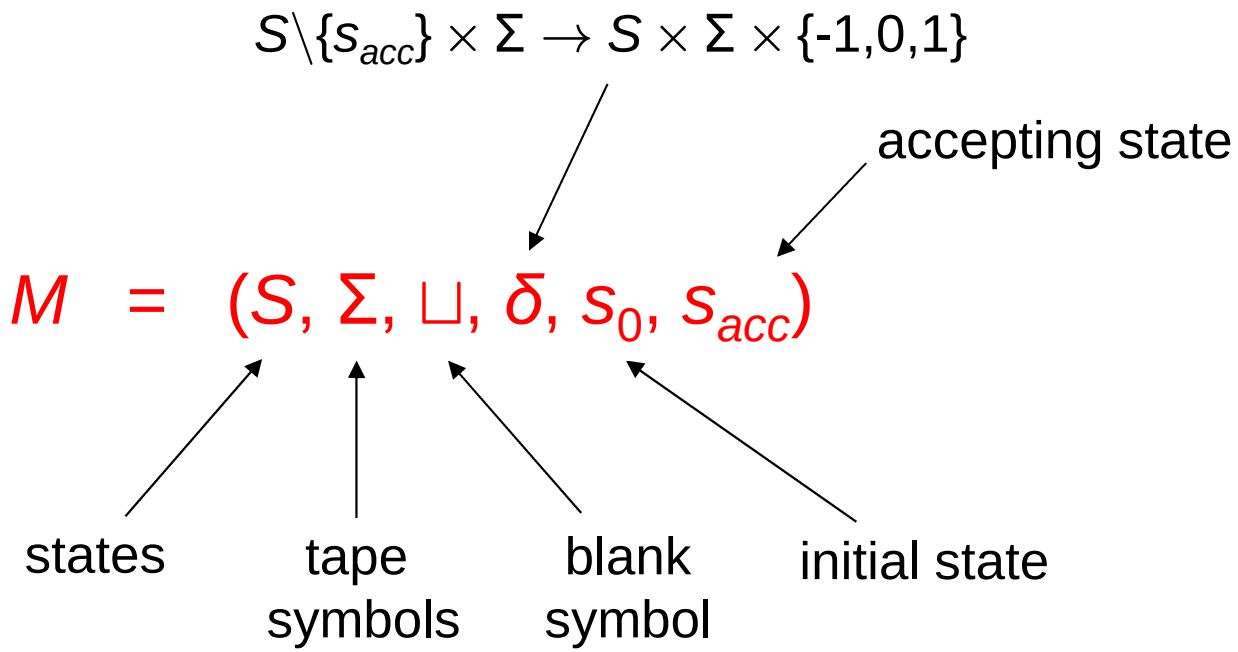
Combined Complexity of Datalog

Theorem: Datalog is EXPTIME-complete in combined complexity

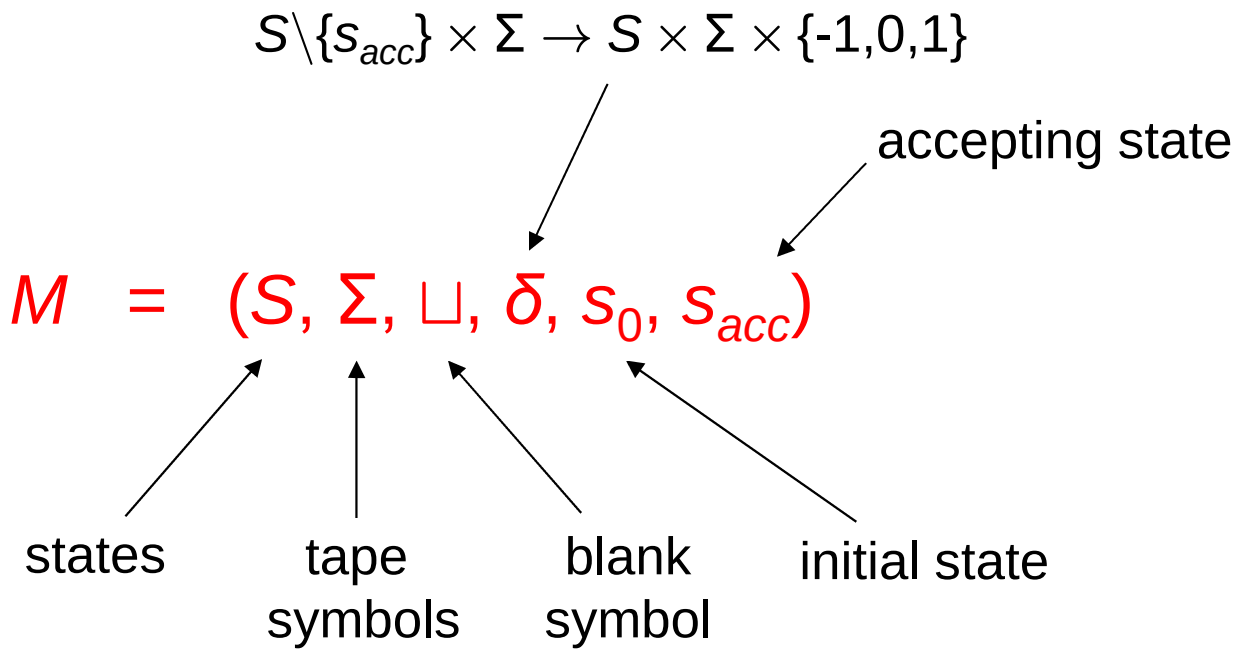
Proof (EXPTIME-hard):

by simulating an EXPTIME Turing machine

Deterministic Turing Machine (DTM)



Deterministic Turing Machine (DTM)



$\delta(s_1,a) = (s_2,b,1)$

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

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

EXPTIME-hardness of Datalog

The goal: encode the EXPTIME computation of a DTM M on input string I with a Datalog program P, a database D, and an atom α such that $\alpha \in P(D)$ iff M accepts I in at most $N = 2^m$ steps, where $m = |I|^k$

The Relational Schema

Time points and tape positions from 0 to N-1, are encoded using m-ary tuples $\{0,1\}^m$ (recall that $N = 2^m$) such that 0 = (0,…,0), 1 = (0,…,1), …, N-1 = (1,…,1)

- *Symbol*[*a*](**T**,**C**) - at time instant **T**, cell **C** contains *a*
- *Cursor*(**T**,**C**) - at time instant **T**, cursor points to cell **C**
- *State*[*s*](**T**) - at time instant **T**, the machine is in state *s*
- *Accept*(**T**) - at time instant **T**, the machine accepts

where **T** = $T_1,\ldots,T_m$ and **C** = $C_1,\ldots,C_m$

Initialization Rules

- Assume that $I = a_1 \ldots a_n$
- Assume that we have the relations $First_m$, $Succ_m$ and $\prec_m$ (will be defined later)

the number i

$$Symbol[a_i](\mathbf{T},\mathbf{t_i}) \leftarrow First_m(\mathbf{T})$$

$$Cursor(\mathbf{T},\mathbf{T}) \leftarrow First_m(\mathbf{T})$$

$$State[s_0](\mathbf{T}) \leftarrow First_m(\mathbf{T})$$

$$Symbol[\sqcup](\mathbf{T},\mathbf{C}) \leftarrow First_m(\mathbf{T}),\ \prec_m(\mathbf{t},\mathbf{C})$$

the number n

Transition Rules

$$\delta(s_1,a) \;=\; (s_2,b,1)$$

$$\mathit{Symbol}[b](\mathbf{T}',\mathbf{C}) \leftarrow \mathit{State}[s_1](\mathbf{T}),\ \mathit{Symbol}[a](\mathbf{T},\mathbf{C}),\ \mathit{Cursor}(\mathbf{T},\mathbf{C}),\ \mathit{Succ}_m(\mathbf{T},\mathbf{T}')$$

$$\mathit{Cursor}(\mathbf{T}',\mathbf{C}') \leftarrow \mathit{State}[s_1](\mathbf{T}),\ \mathit{Symbol}[a](\mathbf{T},\mathbf{C}),\ \mathit{Cursor}(\mathbf{T},\mathbf{C}),\ \mathit{Succ}_m(\mathbf{T},\mathbf{T}'),\ \mathit{Succ}_m(\mathbf{C},\mathbf{C}')$$

$$\mathit{State}[s_2](\mathbf{T}') \leftarrow \mathit{State}[s_1](\mathbf{T}),\ \mathit{Symbol}[a](\mathbf{T},\mathbf{C}),\ \mathit{Cursor}(\mathbf{T},\mathbf{C}),\ \mathit{Succ}_m(\mathbf{T},\mathbf{T}')$$

Inertia Rules

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

$$\mathit{Symbol}[a](\mathbf{T}',\mathbf{C}) \leftarrow \mathit{Symbol}[a](\mathbf{T},\mathbf{C}), \mathit{Cursor}(\mathbf{T},\mathbf{C}'), \prec_m(\mathbf{C},\mathbf{C}'), \mathit{Succ}_m(\mathbf{T},\mathbf{T}')$$

$$\mathit{Symbol}[a](\mathbf{T}',\mathbf{C}) \leftarrow \mathit{Symbol}[a](\mathbf{T},\mathbf{C}), \mathit{Cursor}(\mathbf{T},\mathbf{C}'), \prec_m(\mathbf{C}',\mathbf{C}), \mathit{Succ}_m(\mathbf{T},\mathbf{T}')$$

Accepting Rule

Once we reach the accepting state we accept

$$\mathit{Accept} \leftarrow \mathit{State}[s_{acc}](\mathbf{T})$$

Defining $First_m$, $Succ_m$ and $\prec_m$

We assume that $D = \{First_0(0), Last_1(1), Succ_1(0,1)\}$

$Z \in \{0,1\}$

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

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

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

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

inductive definition of $First_{i+1}$ and $Succ_{i+1}$

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

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

definition of $\prec_m$

Concluding EXPTIME-hardness of Datalog

– Several rules but polynomially many $\Rightarrow$ feasible in PTIME

– *Accept* $\in$ *P*(*D*) iff *M* accepts *I* in at most *N* steps

– Can be formally shown by induction on the time steps

Datalog as an Ontology Language

– Ontology languages are usually based on description logics (prev. lecture)

– Much is possible with Datalog

DL Axiom	Datalog Rule
$Parent \sqcap Male \sqsubseteq Father$	$Father(X) \leftarrow Parent(X), Male(X)$
$MetalDevice \sqsubseteq \forall hasPart.Metal$	$Metal(Y) \leftarrow MetalDevice(X), hasPart(X,Y)$
$brotherOf \sqsubseteq relativeOf$	$relativeOf(X,Y) \leftarrow brotherOf(X,Y)$
$parentOf$ **inv** $childOf$	$childOf(Y,X) \leftrightarrow parentOf(X,Y)$
trans($ancestorOf$)	$ancOf(X,Z) \leftarrow ancOf(X,Y), ancOf(Y,Z)$
$SeniorEmp \times Emp \sqsubseteq moreThan$	$moreThan(X,Y) \leftarrow SeniorEmp(X), Emp(Y)$

Datalog as an Ontology Language

– Ontology languages are usually based on description logics (prev. lecture)

– Much is not possible with Datalog

[Patel-Schneider & Horrocks, Journal of Web Semantics 2007]

DL Axiom	?
$Employee \sqsubseteq \exists reportsTo$	$\exists Y\ reportsTo(X,Y) \leftarrow Employee(X)$
funct(*reportsTo*)	$Y = Z \leftarrow reportsTo(X,Y), reportsTo(X,Z)$
Employee **disj** *Customer*	$\bot \leftarrow Employee(X), Customer(X)$

Datalog$^{\pm}$

– Extend Datalog by allowing in the head:

  - Existential quantification ($\exists$)

  - Equality atoms ($=$)

  - Constant false ($\bot$)

} Datalog[$\exists$,=,$\bot$]

– As we shall see, Datalog[$\exists$] is undecidable

– Datalog[$\exists$,=,$\bot$] is syntactically restricted $\rightarrow$ **Datalog$^{\pm}$**

Datalog Extensions

- Formal Syntax of Datalog[∃]
- Fixpoint Semantics of Datalog[∃]
- Undecidability of Datalog[∃]
- Guardedness, Linearity and Stickiness
- Additional Features (=,⊥)

Syntax of Datalog[∃]

A Datalog[∃] rule is an expression

$$\underbrace{\exists \mathbf{Y}\, R_0(\mathbf{X_0}, \mathbf{Y})}_{\text{head}} \leftarrow \underbrace{R_1(\mathbf{X_1}), \ldots, R_n(\mathbf{X_n})}_{\text{body}}$$

- $n \geq 0$ (empty body)
- $R_0, \ldots, R_n$ are relation symbols (or predicates)
- $\mathbf{X_0}, \ldots, \mathbf{X_n}$ are tuples of terms (constants or variables)
- $\mathbf{Y}$ is a tuple of variables (disjoint from $\mathbf{X_0} \cup \ldots \cup \mathbf{X_n}$)

Syntax of Datalog[∃]

- Datalog[∃] program *P* - a (finite) set of Datalog[∃] rules
- *sch*(*P*) - the set of predicates ocurring in *P*
- Note: *sch*(*P*) is no longer partitioned into *idb*(*P*) and *edb*(*P*) - why?

Syntax of Datalog[∃]

- Datalog[∃] program P - a (finite) set of Datalog[∃] rules
- $sch(P)$ - the set of predicates ocurring in P
- Note: $sch(P)$ is no longer partitioned into $idb(P)$ and $edb(P)$ - why?

DL Ontology	Datalog[∃] Program P
$Person \sqsubseteq \exists hasFather$	$\exists Y\ hasFather(X,Y) \leftarrow Person(X)$
$\exists hasFather^- \sqsubseteq Person$	$Person(Y) \leftarrow hasFather(X,Y)$

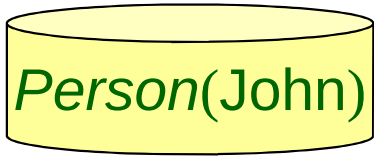
All predicates of $sch(P)$ appear in the body and in the head

Fixpoint Semantics

Analogous to the fixpoint semantics of Datalog - chase procedure

Input: Database D, Datalog[$\exists$] program P

Output: Instance for $sch(P)$ that satisfies P



$\exists Y\ hasFather(X,Y) \leftarrow Person(X)$ $\quad$ $Person(Y) \leftarrow hasFather(X,Y)$

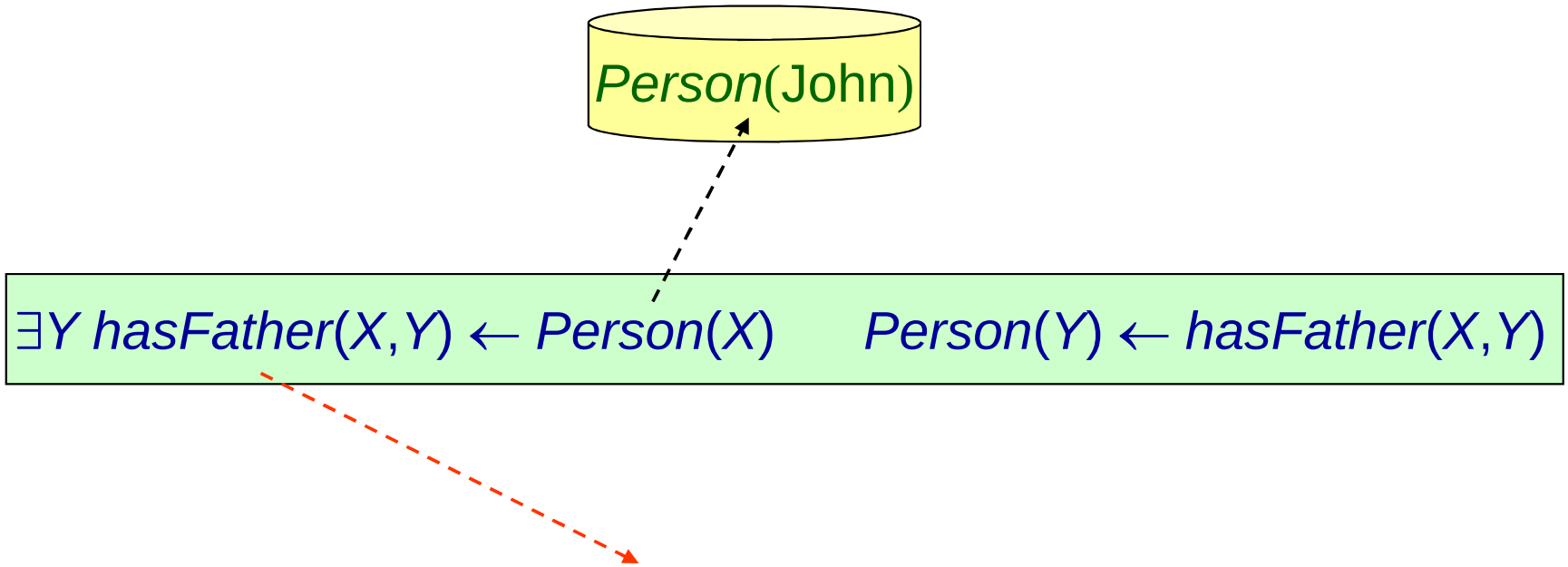
$chase(D,P) = D \cup ?$

Fixpoint Semantics

Analogous to the fixpoint semantics of Datalog - chase procedure

Input: Database D, Datalog[$\exists$] program P

Output: Instance for $sch(P)$ that satisfies P



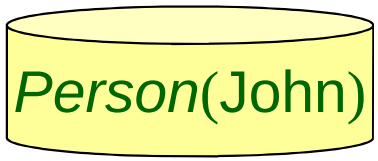
$chase(D,P) = D \cup \{hasFather(\text{John}, z_1)$

Fixpoint Semantics

Analogous to the fixpoint semantics of Datalog - chase procedure

Input: Database D, Datalog[$\exists$] program P

Output: Instance for $sch(P)$ that satisfies P



$\exists Y\ hasFather(X,Y) \leftarrow Person(X)$ $\quad$ $Person(Y) \leftarrow hasFather(X,Y)$

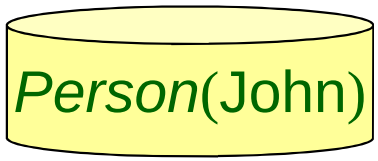
$chase(D,P) = D \cup \{hasFather(\text{John}, z_1), Person(z_1)\}$

Fixpoint Semantics

Analogous to the fixpoint semantics of Datalog - chase procedure

Input: Database D, Datalog[$\exists$] program P

Output: Instance for $sch(P)$ that satisfies P



$\exists Y\ hasFather(X,Y) \leftarrow Person(X)$ $\quad$ $Person(Y) \leftarrow hasFather(X,Y)$

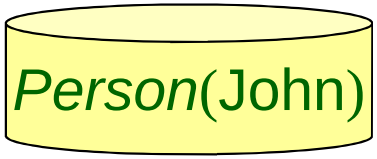
$chase(D,P) = D \cup \{hasFather(\text{John},z_1), Person(z_1), hasFather(z_1,z_2)$

Fixpoint Semantics

Analogous to the fixpoint semantics of Datalog - chase procedure

Input: Database D, Datalog[$\exists$] program P

Output: Instance for $sch(P)$ that satisfies P



$\exists Y\ hasFather(X,Y) \leftarrow Person(X)$ $\quad$ $Person(Y) \leftarrow hasFather(X,Y)$

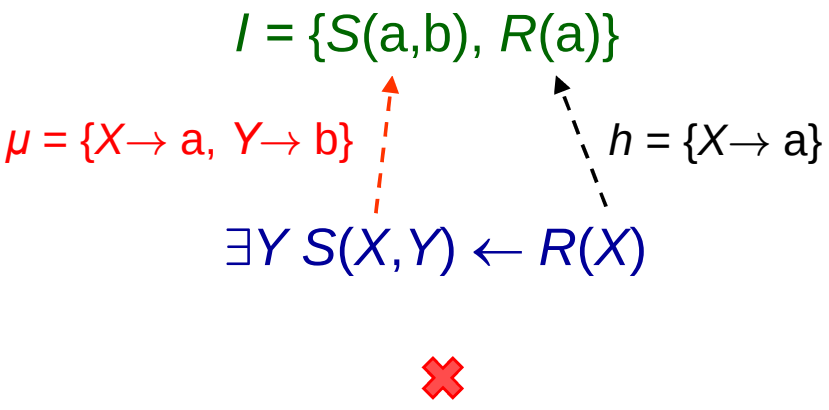
$chase(D,P) = D \cup \{hasFather(\text{John},z_1), Person(z_1), hasFather(z_1,z_2) \ \cdots$

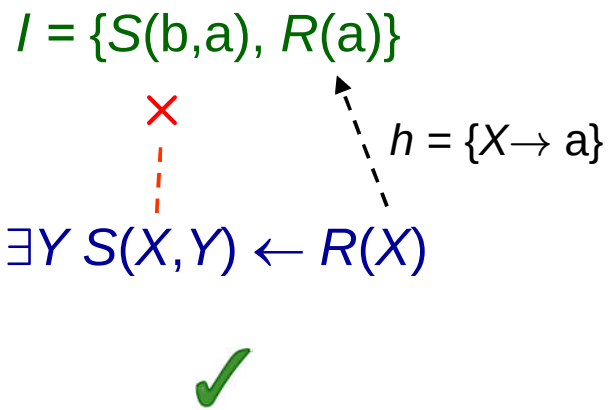
Fixpoint Semantics

– Chase rule - the building block of the chase procedure

– A rule $\rho = \exists \mathbf{Y}\ R_0(\mathbf{X_0},\mathbf{Y}) \leftarrow R_1(\mathbf{X_1}),\ldots,R_n(\mathbf{X_n})$ is applicable to instance I if:

- exists homomorphism h such that $\{h(R_1(\mathbf{X_1})),\ldots,h(R_n(\mathbf{X_n}))\} \subseteq I$
- but no $\mu \supseteq h$ such that $\mu(R_0(\mathbf{X_0},\mathbf{Y})) \in I$





Fixpoint Semantics

– Chase rule - the building block of the chase procedure

– A rule $\rho = \exists \mathbf{Y}\ R_0(\mathbf{X_0}, \mathbf{Y}) \leftarrow R_1(\mathbf{X_1}), \ldots, R_n(\mathbf{X_n})$ is applicable to instance I if:

- exists homomorphism h such that $\{h(R_1(\mathbf{X_1})), \ldots, h(R_n(\mathbf{X_n}))\} \subseteq I$

- but no $\mu \supseteq h$ such that $\mu(R_0(\mathbf{X_0}, \mathbf{Y})) \in I$

– Let $J = I \cup \{\mu(R_0(\mathbf{X_0}, \mathbf{Y}))\}$, where $\mu \supseteq h$ and $\mu(Y_i)$ is a fresh value not in I

– The result of applying ρ to I is J, denoted $I \xrightarrow{\rho,h} J$ - chase step

Fixpoint Semantics

– A finite chase of D w.r.t. to P is a finite sequence

$$D \xrightarrow{\rho_1,h_1} I_1 \xrightarrow{\rho_2,h_2} I_2 \xrightarrow{\rho_3,h_3} \cdots \xrightarrow{\rho_m,h_m} I_m$$

and *chase*(D,P) is defined as the instance I_m

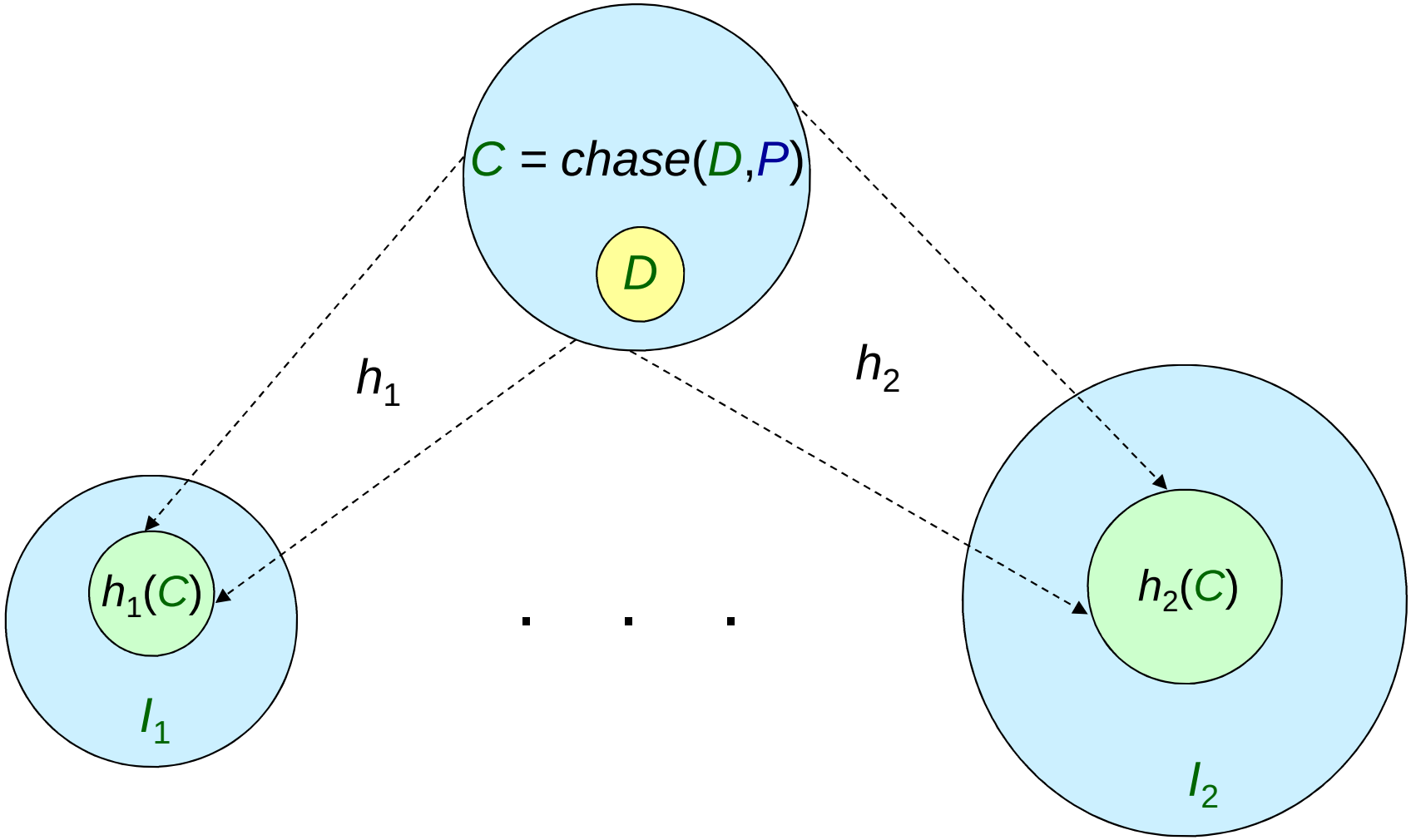
– An infinite chase of D w.r.t. to P is an infinite sequence

$$D \xrightarrow{\rho_1,h_1} I_1 \xrightarrow{\rho_2,h_2} I_2 \xrightarrow{\rho_3,h_3} \cdots \xrightarrow{\rho_m,h_m} I_m \cdots$$

and *chase*(D,P) is defined as the instance $\cup_{j \geq 0} I_j$ (with $I_0 = D$)

– The semantics of P on D, denoted $P(D)$, is defined as *chase*(D,P)

Chase: A Universal Model



$\forall I$ (I model of D and P $\Rightarrow$ $chase(D,P) \xrightarrow{\text{hom}} I$)

Implicit in [Fagin, Kolaitis, Miller & Popa, Theoretical Computer Science 2005]

Chase: Uniqueness Property

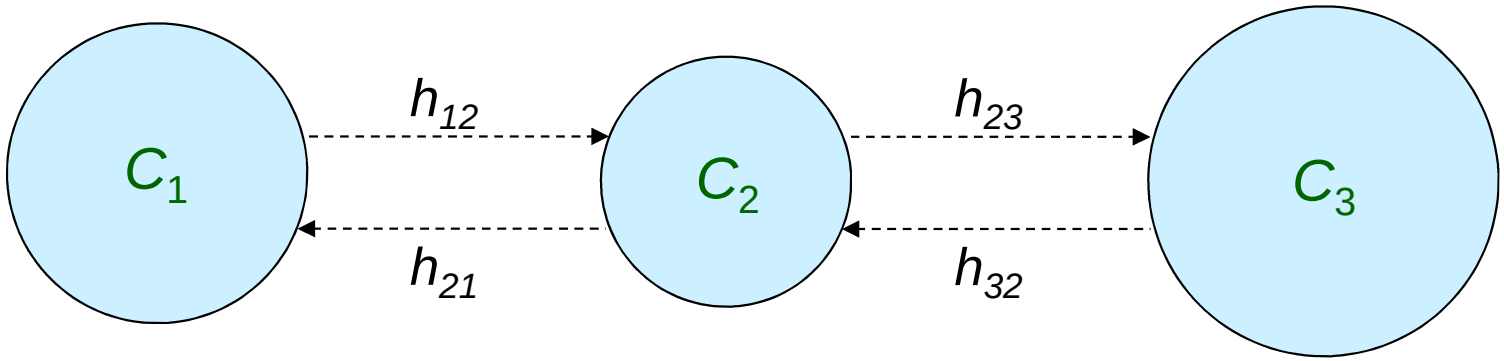
– In general is not unique - depends on the order of rule application

$D = \{R(a)\}$ $\rho_1 = \exists Y\, S(Y) \leftarrow R(X)$ $\rho_2 = S(X) \leftarrow R(X)$

$\text{Solution}_1 = \{R(a), S(z), S(a)\}$ ρ_1 then ρ_2

$\text{Solution}_2 = \{R(a), S(a)\}$ ρ_2 then ρ_1

– Unique up to homomorphic equivalence



Chase: The Challenge of Infinity

– In general is infinite

$D = \{R(a,b)\}$ $\quad\quad$ $\exists Z\, R(Y,Z) \leftarrow R(X,Y)$

Solution = $\{R(a,b), R(b,z_1), R(z_1,z_2), R(z_2,z_3), \ldots\}$

– For plain Datalog, the fixpoint semantics provides an algorithm

– The situation changes dramatically for Datalog[$\exists$] - undecidable

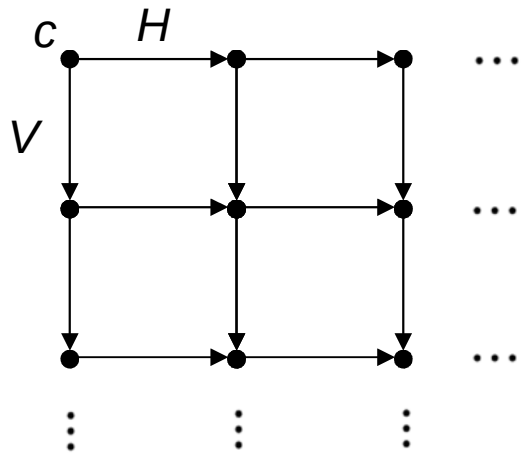
Undecidability of Datalog[∃]

Theorem: Datalog[∃] is undecidable

Proof :

by simulating a deterministic Turing machine with an empty tape

Build an Infinite Grid



i-th horizontal line represents the

i-th configuration of the machine

Start(*c*) fixes
the starting point

$$Node(X) \leftarrow Start(X)$$

$$Initial(X) \leftarrow Start(X)$$

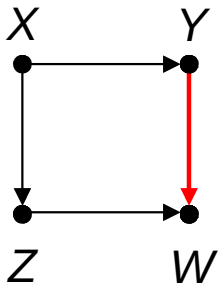
$$\exists Y\ H(X,Y) \leftarrow Node(X)$$

$$Node(Y) \leftarrow H(X,Y)$$

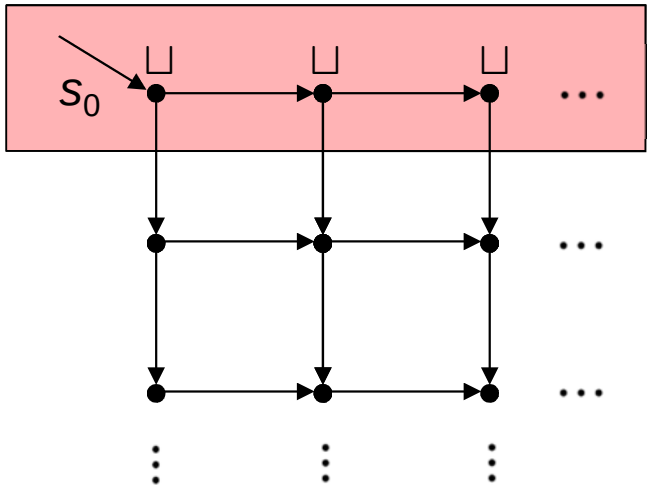
$$\exists Y\ V(X,Y) \leftarrow Node(X)$$

$$Node(Y) \leftarrow V(X,Y)$$

$$V(Y,W) \leftarrow H(X,Y), H(Z,W), V(X,Z)$$



Initialization Rules

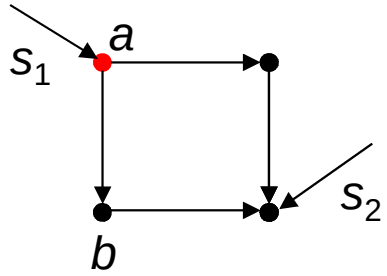


$$\mathit{Initial}(Y) \leftarrow \mathit{Initial}(X), H(X,Y)$$

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

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

Transition Rules



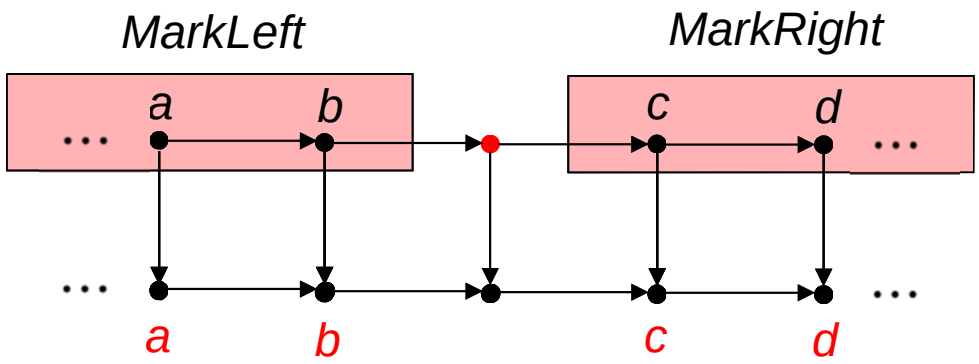
$$\delta(s_1,a) = (s_2,b,1)$$

$$Cursor[s_2](Z) \leftarrow Cursor[s_1](X), Symbol[a](X), V(X,Y), H(Y,Z)$$

$$Symbol[b](Y) \leftarrow Cursor[s_1](X), Symbol[a](X), V(X,Y)$$

$$Mark(X) \leftarrow Cursor[s_1](X), Symbol[a](X)$$

Inertia Rules



$MarkRight(Y) \leftarrow Mark(X), H(X,Y)$

$MarkRight(Y) \leftarrow MarkRight(X), H(X,Y)$

$Symbol[a](Y) \leftarrow MarkRight(X), Symbol[a](X), V(X,Y)$

We need similar rules for the cells before the cursor

Accepting Rule

Once we reach the accepting state we accept

$$Accept \leftarrow Cursor[s_{acc}](X)$$

$Accept \in P(D)$ iff the DTM accepts

Undecidability of Datalog[∃]

Theorem: Datalog[∃] is undecidable

Proof :

by simulating a deterministic Turing machine with an empty tape

… syntactic restrictions are needed!

Guarded Datalog[∃]

– Inspired by the guarded fragment of first-order logic

[Andréka, van Benthem & Németi, Journal of Philosophical Logic 98]

– There exists a body-atom that contains all the body-variables - guard

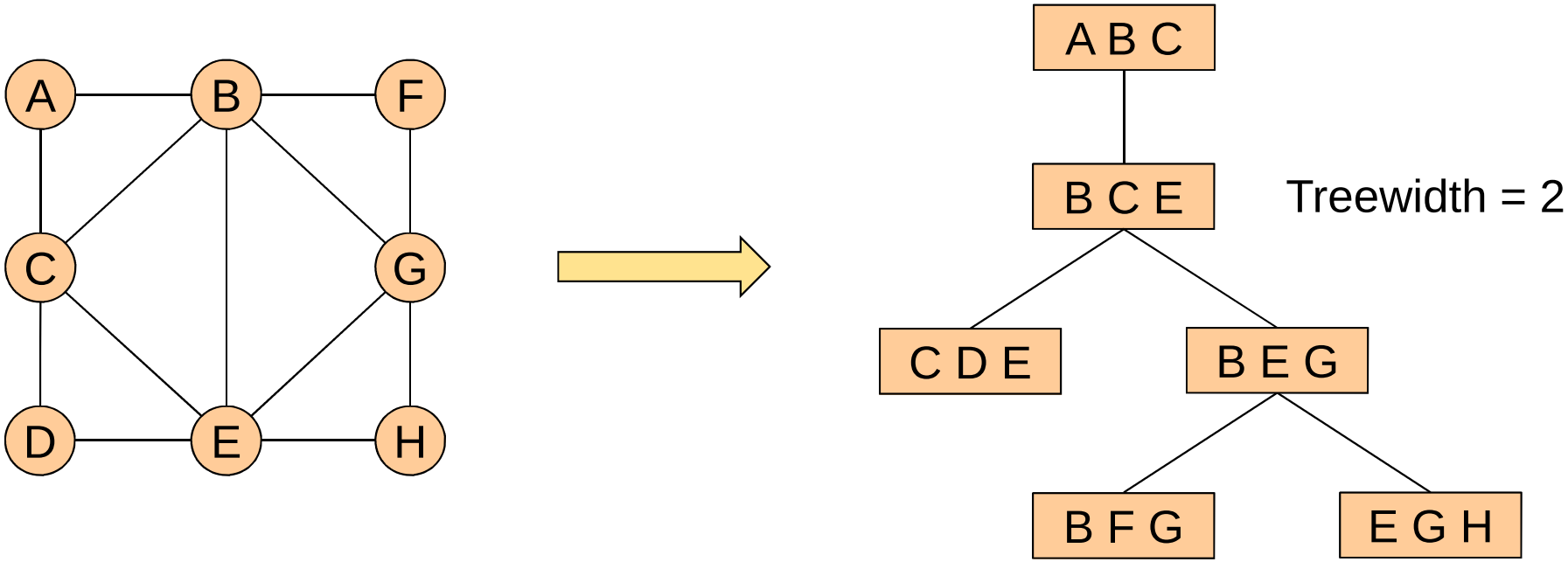
$$Manager(X) \leftarrow Employee(X), supervisorOf(X,Y), Manager(Y)$$

– Chase has finite treewidth $\Rightarrow$ Guarded Datalog[∃] is decidable

[Calì, Gottlob & Kifer, KR 2008]

Treewidth of the Chase

– Tree decomposition - a mapping of a graph into a tree



– Treewidth - number of graph vertices mapped to any treenode (in fact, -1)

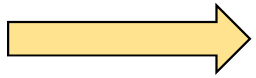
Treewidth of the Chase

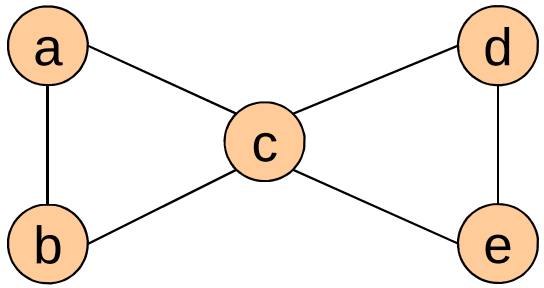
– An instance *I* can be represented as a graph - Gaifman graph

R(a,b,c)

S(c,d)

T(c,d,e)





– Treewidth of *I* is defined as the treewidth of its Gaifman graph

– Chase has finite treewidth $\Rightarrow$ is a tree-like structure

Guarded Datalog[∃]

– Inspired by the guarded fragment of first-order logic

[Andréka, van Benthem & Németi, Journal of Philosophical Logic 98]

– There exists a body-atom that contains all the body-variables - guard

$$Manager(X) \leftarrow Employee(X), supervisorOf(X,Y), Manager(Y)$$

– Chase has finite treewidth $\Rightarrow$ Guarded Datalog[∃] is decidable

[Calì, Gottlob & Kifer, KR 08]

– What about the complexity of Guarded Datalog[∃]? - now we consider ontological query answering (previous lecture)

Complexity of Guarded Datalog[∃]

– Ontological query answering problem:

- **Input**: program P, database D for $sch(P)$, conjunctive query Q

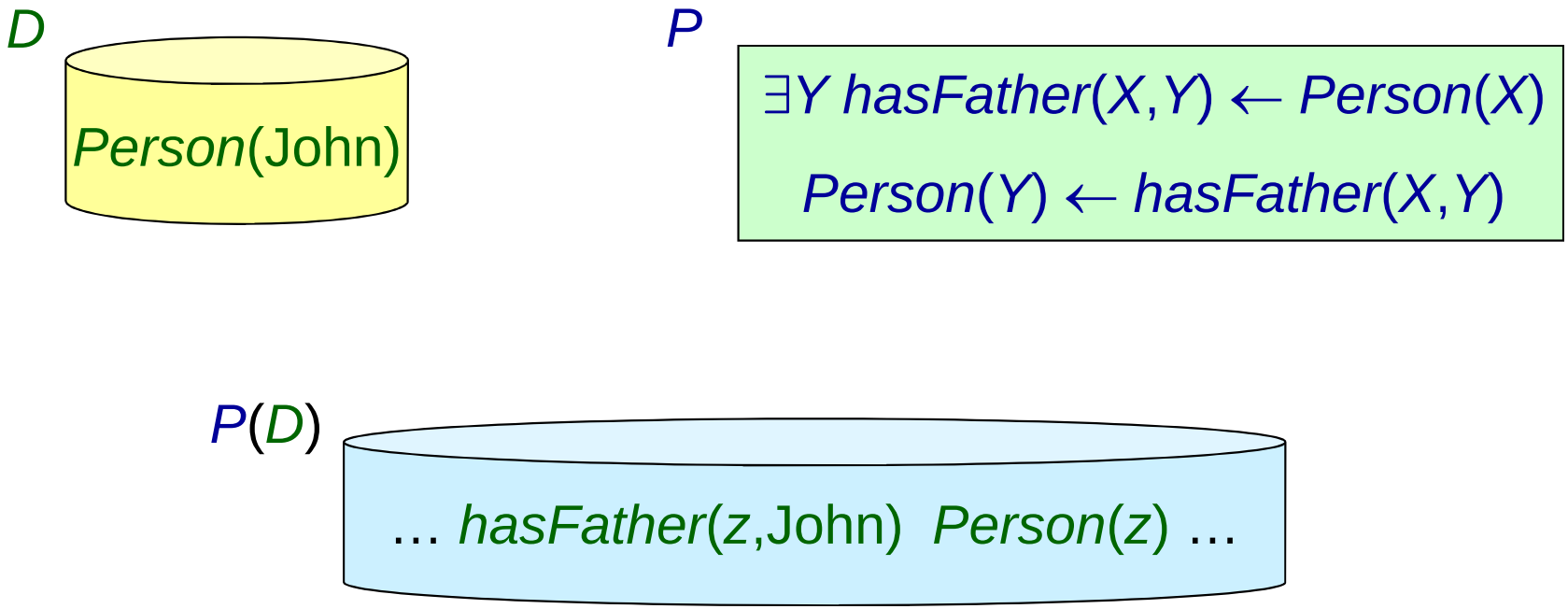
TBox ABox

- **Question**: $P \cup D \models Q$, or, equivalently, $P(D) \models Q$?

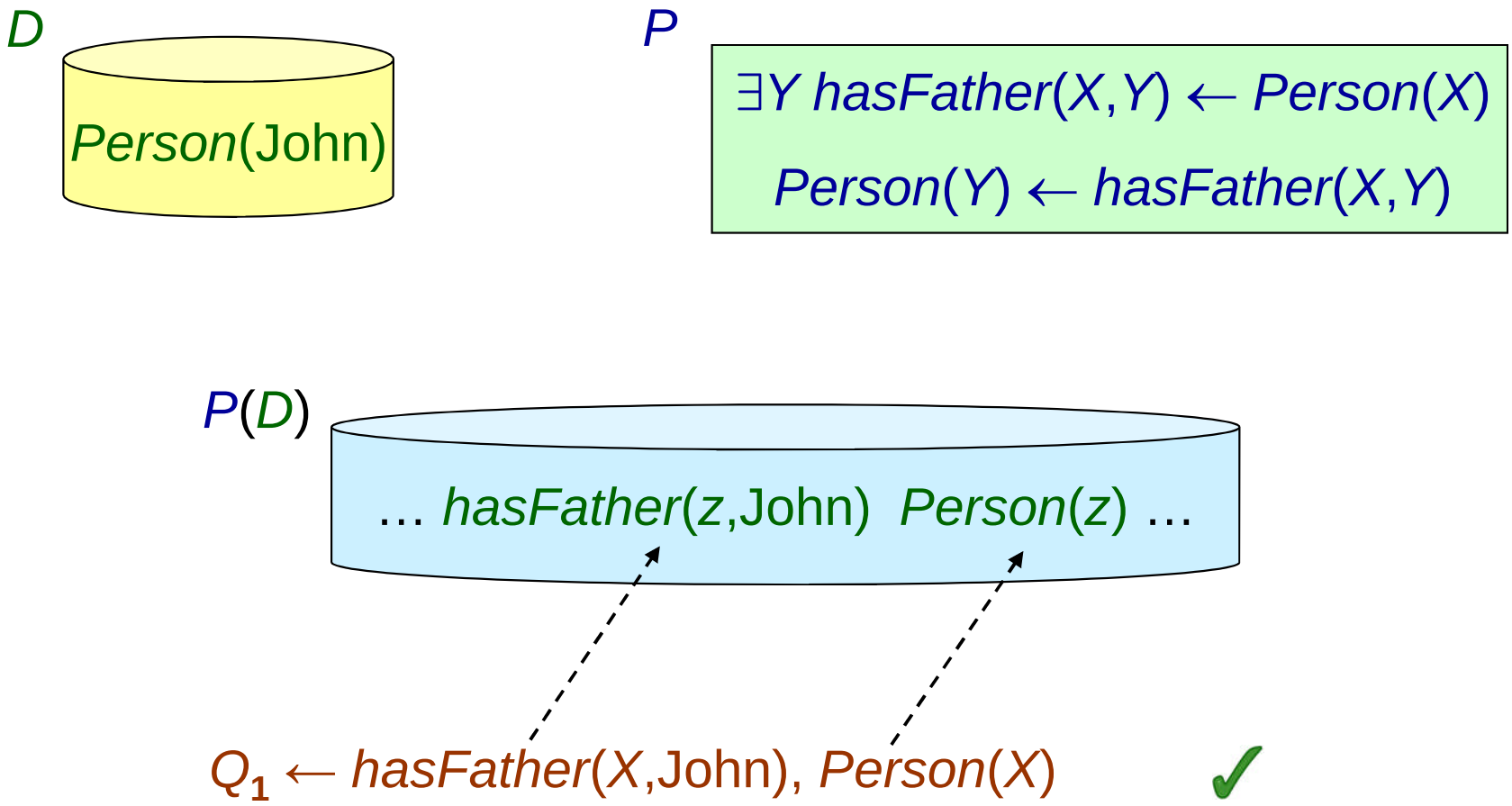
– Data complexity - P and Q fixed, D part of the input

– Combined complexity - everything part of the input

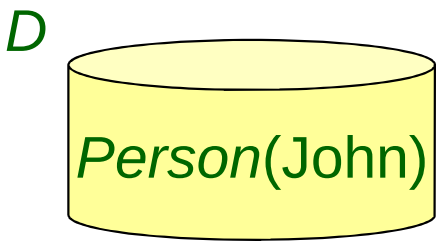
Ontological Query Answering: Example

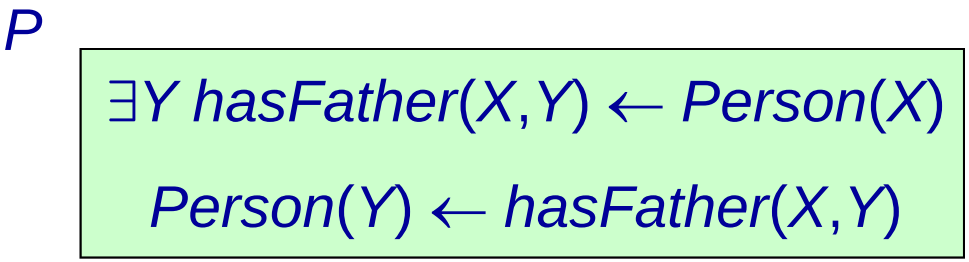


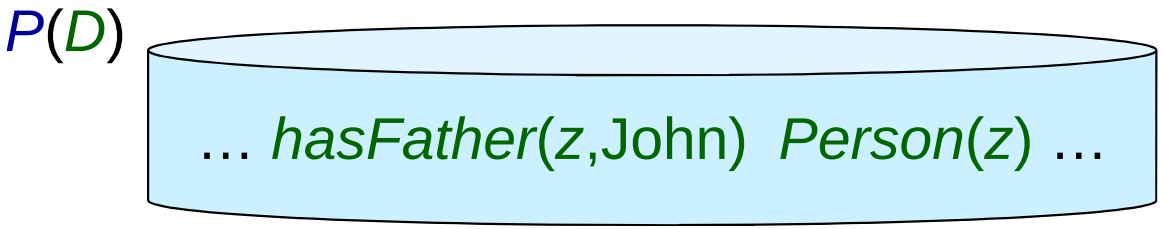
Ontological Query Answering: Example



Ontological Query Answering: Example







$Q_1 \leftarrow hasFather(X,\text{John}), Person(X)$ ✓

$Q_2 \leftarrow hasFather(\text{John},X)$ ✗

Data Complexity of Guarded Datalog[∃]

Theorem: Guarded Datalog[∃] is PTIME-complete in data complexity

Proof (in PTIME):

- Guarded Datalog[∃] enjoys the bounded guard-depth property
- Construct in PTIME the finite part *C* of the guarded chase forest
- Evaluate the given query over *C*

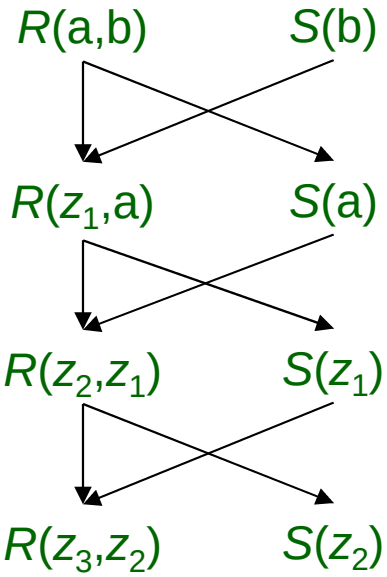
[Calì, Gottlob & Lukasiewicz, Journal of Web Semantics 2012]

Bounded Guard-Depth Property

– Chase graph

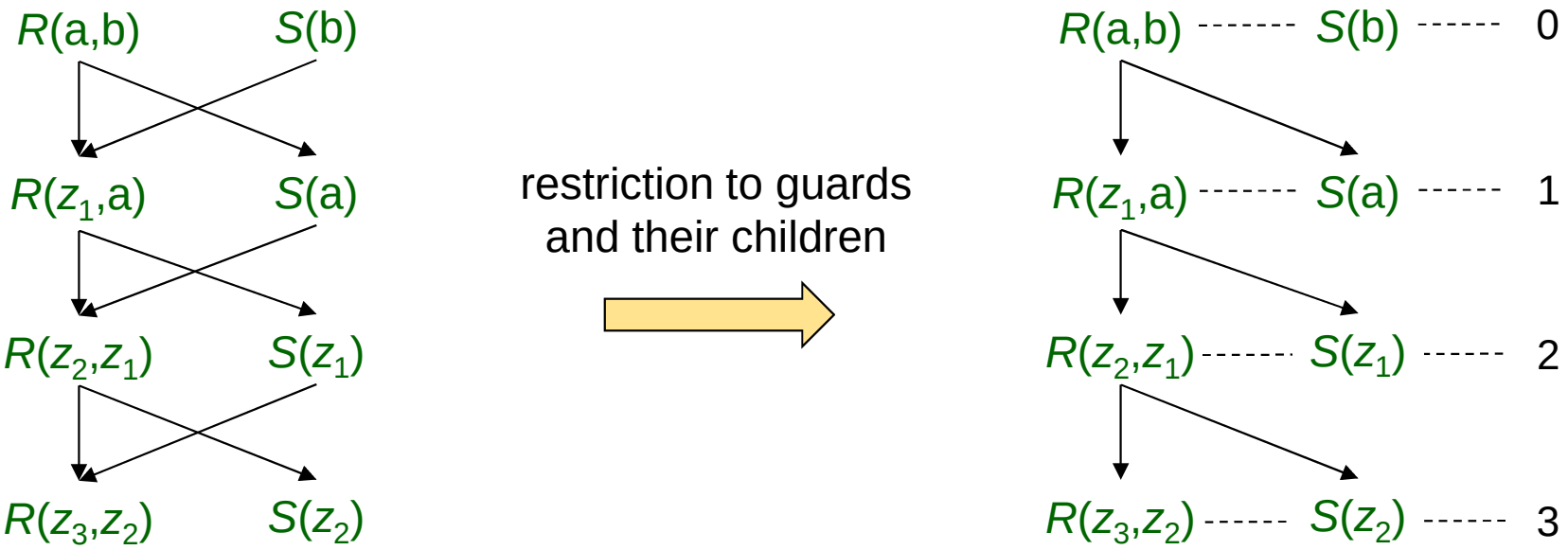
$D = \{R(a,b), S(b)\}$

$$P = \begin{cases} \rho_1 = \exists Z\, R(Z,X) \leftarrow R(X,Y), S(Y) \\ \rho_2 = S(X) \leftarrow R(X,Y) \end{cases}$$

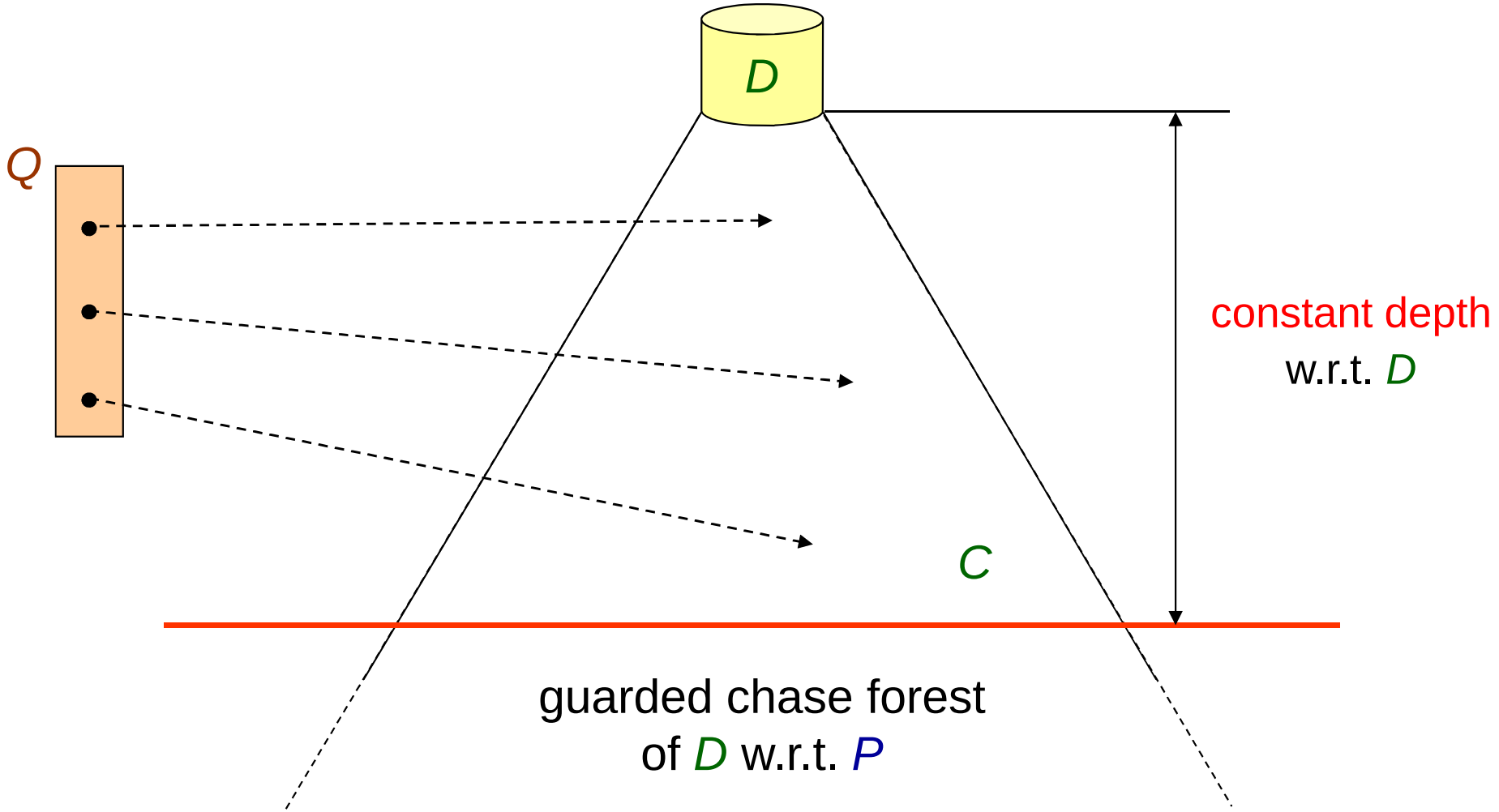


Bounded Guard-Depth Property

– Guarded chase forest



Bounded Guard-Depth Property



$P(D) \models Q \quad \Rightarrow \quad C \models Q$

Data Complexity of Guarded Datalog[∃]

Theorem: Guarded Datalog[∃] is PTIME-complete in data complexity

Proof (in PTIME):

- Guarded Datalog[∃] enjoys the bounded guard-depth property
- Construct in PTIME the finite part *C* of the guarded chase forest
- Evaluate the given query over *C*

[Calì, Gottlob & Lukasiewicz, Journal of Web Semantics 2012]

Data Complexity of Datalog

Theorem: Datalog[∃] is PTIME-complete in data complexity

Proof (PTIME-hard): reduction from fact inference for propositional LP(2)

– For each $R_0 \leftarrow$ add in D the atom $True(R_0)$
– For each $R_0 \leftarrow R_1, R_2$ add in D the atom $S(R_0, R_1, R_2)$

} encode program P in database D

– Construct the fixed Guarded Datalog[∃] program P_{DAT}:

$$T(X) \leftarrow True(X)$$
$$T(Z) \leftarrow T(X), T(Y), S(Z, X, Y)$$

} meta-interpreter for LP(2)

– $P \models Q$ iff $T(Q) \in P_{DAT}(D)$

Combined Complexity of Guarded Datalog[∃]

Theorem: Guarded Datalog[∃] is 2EXPTIME-complete in comb. complexity

Proof:

- Upper bound: alternating EXPSPACE algorithm
- Lower bound: by simulating an alternating EXPSPACE TM

[Calì, Gottlob & Kifer, KR 2008]

DLs vs Guarded Datalog[∃]

ELH: Popular DL (for biological applications) with PTIME data complexity

[Baader, IJCAI 2003 and Rosati, DL 2007]

ELH TBox	Datalog[∃] Representation
$A \sqsubseteq B$	$B(X) \leftarrow A(X)$
$A \sqcap B \sqsubseteq C$	$C(X) \leftarrow A(X), B(X)$
$\exists R.A \sqsubseteq B$	$B(X) \leftarrow R(X,Y), A(Y)$
$A \sqsubseteq \exists R.B$	$\exists Y\, Aux(X,Y) \leftarrow A(X) \quad R(X,Y) \leftarrow Aux(X,Y) \quad B(Y) \leftarrow Aux(X,Y)$
$R \sqsubseteq S$	$S(X,Y) \leftarrow R(X,Y)$

DLs vs Guarded Datalog[∃]

DL-Lite: Popular family of DLs with AC_0 data complexity (OWL 2 QL)

[Calvanese, De Giacomo, Lembo, Lenzerini & Rosati, Journal of Automated Reasoning 2007]

DL-Lite$_R$ TBox	Datalog[∃] Representation
$A \sqsubseteq B$	$B(X) \leftarrow A(X)$
$A \sqsubseteq \exists R$	$\exists Y\, R(X,Y) \leftarrow A(X)$
$\exists R \sqsubseteq A$	$A(X) \leftarrow R(X,Y)$
$R \sqsubseteq S$	$S(X,Y) \leftarrow R(X,Y)$

Note: Disjointness assertions are harmless - will be treated differently

Linear Datalog[∃]

- Goal: Lightweight Datalog[∃] fragment which is highly tractable

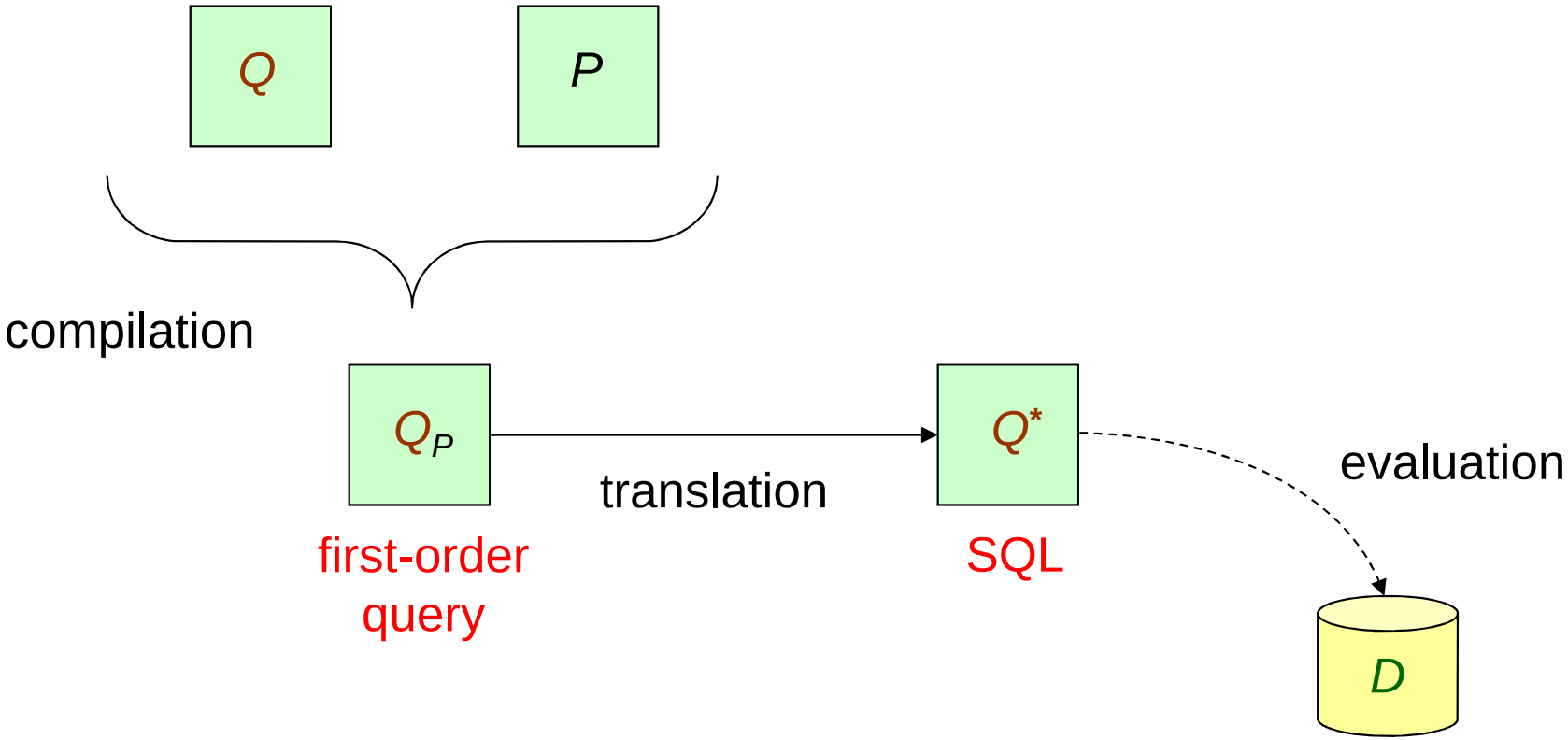
- There exists only one atom in the body

$$\exists Y\ hasFather(X,Y) \leftarrow Person(X)$$

- Enjoys first-order rewritability

  [Calì, Gottlob & Lukasiewicz, Journal of Web Semantics 2012]

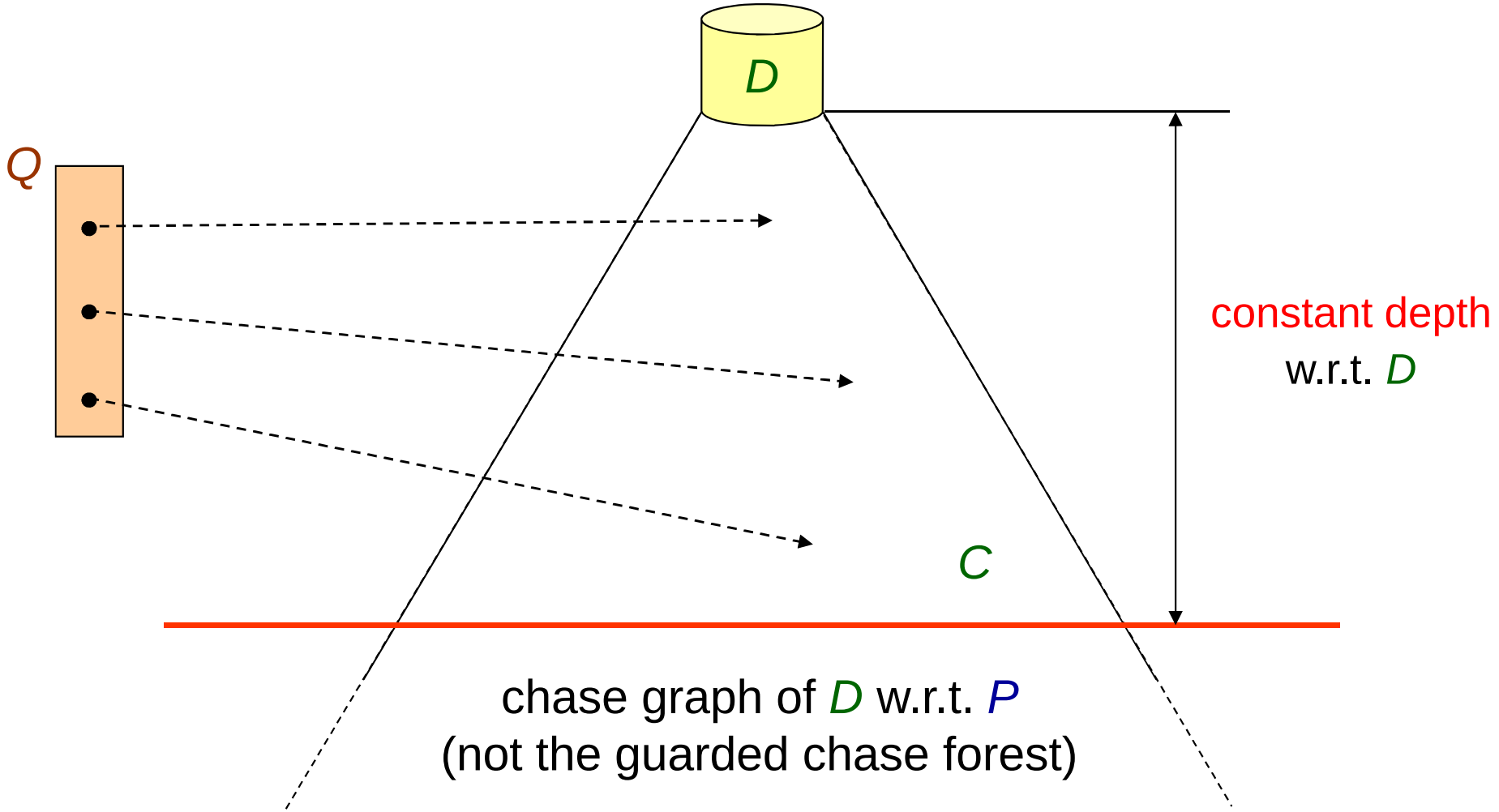
First-Order Rewritability



$$\forall D\ (P(D) \models Q \quad \Leftrightarrow \quad D \models Q^*)$$

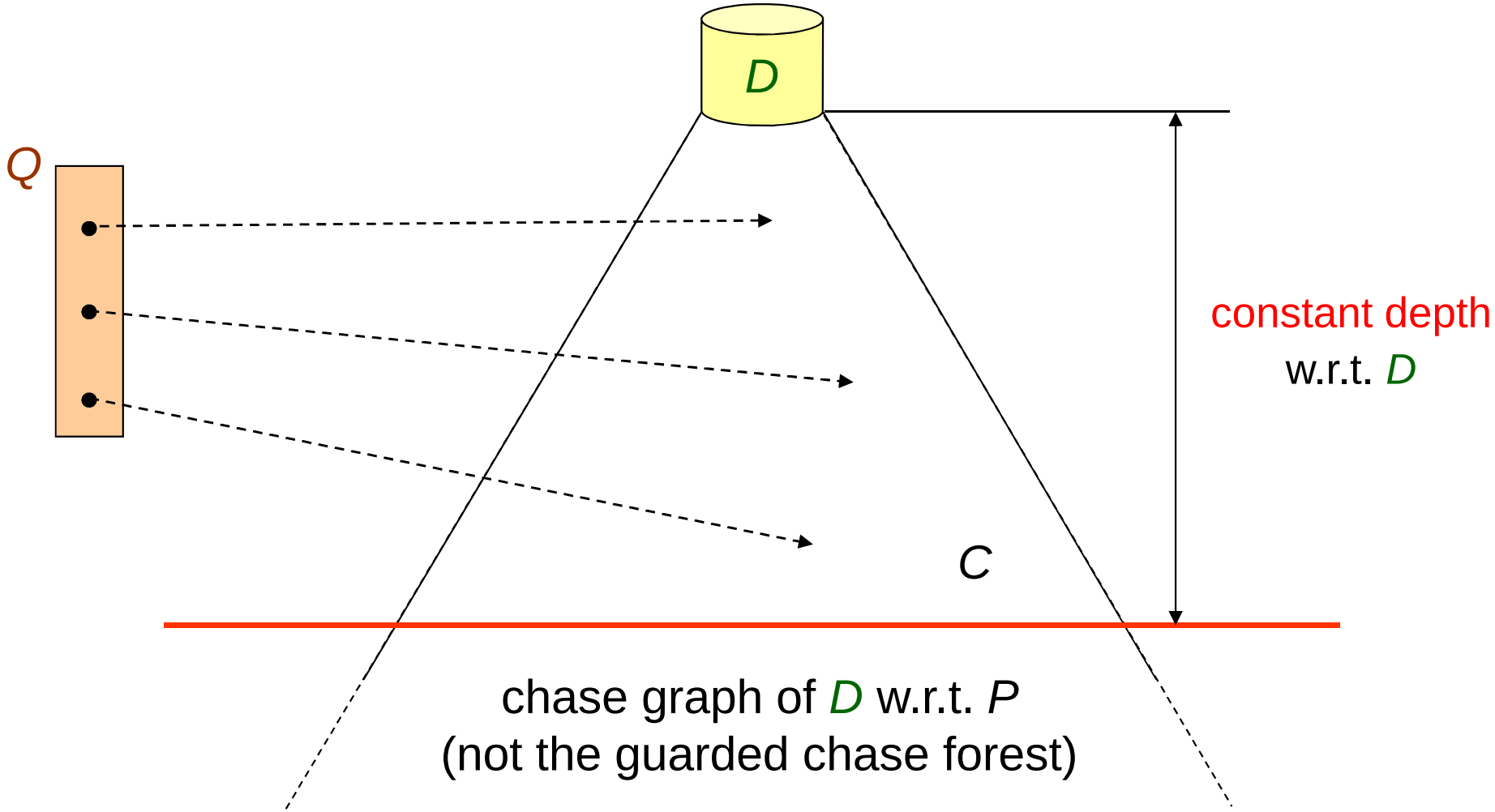
[Calvanese, De Giacomo, Lembo, Lenzerini & Rosati, Journal of Automated Reasonig 2007]

Bounded Derivation-Depth Property (BDDP)



$P(D) \models Q \quad \Rightarrow \quad C \models Q$

Bounded Derivation-Depth Property (BDDP)



Sufficient condition for first-order rewritability

Linear Datalog[∃]

– Goal: Lightweight Datalog[∃] fragment which is highly tractable

– There exists only one atom in the body

$$\exists Y\ hasFather(X,Y) \leftarrow Person(X)$$

– Enjoys first-order rewritability

[Calì, Gottlob & Lukasiewicz, Journal of Web Semantics 2012]

– What about the complexity of Linear Datalog[∃]?

Data Complexity of Linear Datalog[∃]

Theorem: Linear Datalog[∃] is in AC_0 in data complexity

Proof:

- We exploit first-order rewritability of Linear Datalog[∃]
- Construct the first-order rewritten query Q_{FO} (in constant time)
- Evaluate Q_{FO} over the given database (in AC_0)
  [Vardi, PODS 1995]

[Calì, Gottlob & Lukasiewicz, Journal of Web Semantics 2012]

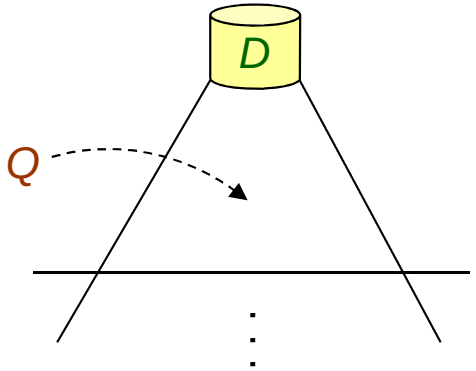
Combined Complexity of Linear Datalog[∃]

Theorem: Linear Datalog[∃] is PSPACE-complete in comb. complexity

Proof:

– Upper bound: exploit the BDDP

implicit in [Johnson & Klug, JCSS 1984]



– Lower bound: by simulating a PSPACE Turing machine

PSPACE-hardness of Linear Datalog[∃]

- Assume that the tape alphabet is {0,1,␣}
- Suppose that M halts on $I = a_1 \ldots a_m$ using $n = m^k$ cells, for $k > 0$
- Initial configuration (the database D)

$$Config(s_{init}, a_1, \ldots, a_m, \underbrace{␣, \ldots, ␣}_{n-m}, 1, \underbrace{0, \ldots, 0}_{n-1})$$

- Transition rule - $\delta(s_1, a) = (s_2, b, 1)$

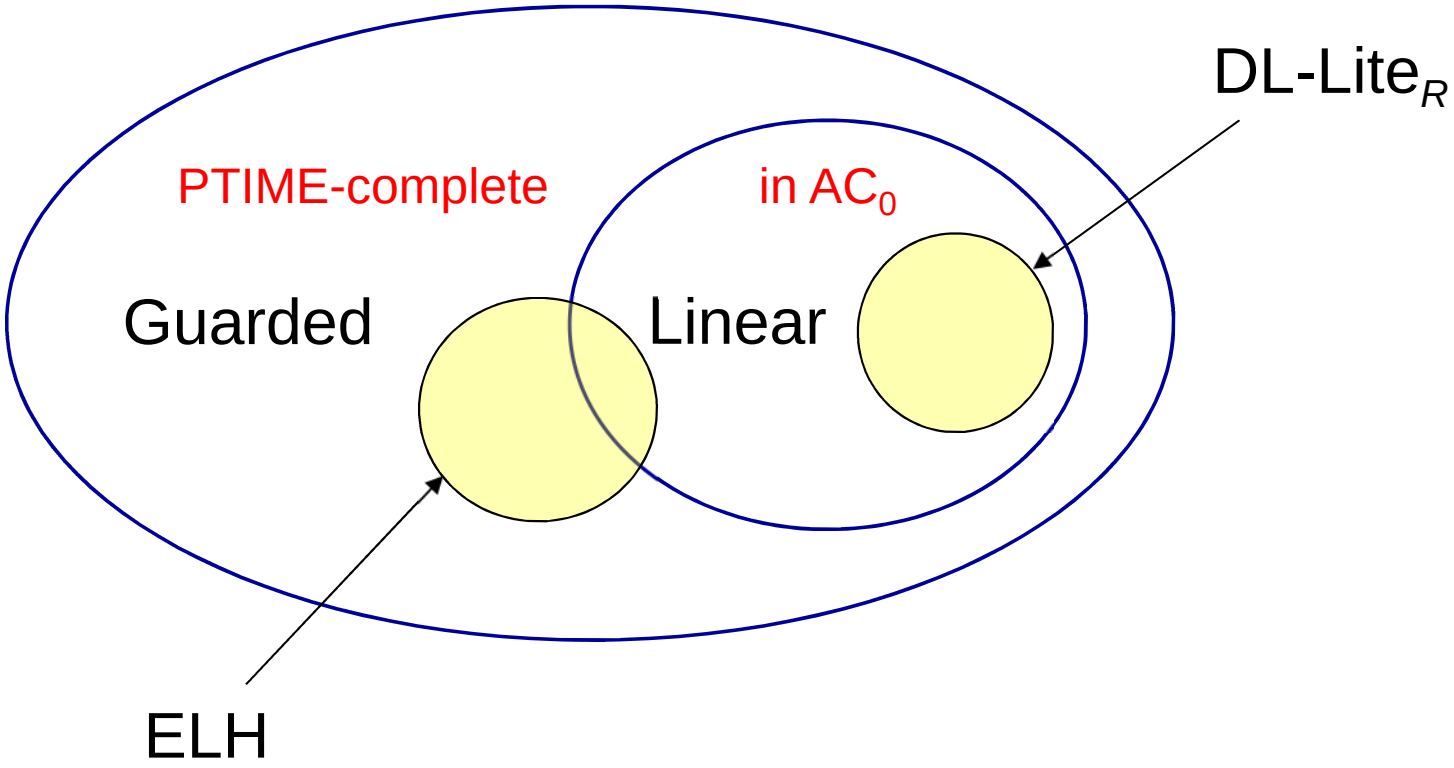
$$Config(s_1, X_1, \ldots, X_{i-1}, a, X_{i+1}, \ldots, X_n, \overbrace{0, \ldots, 0}^{i-1}, 1, \overbrace{0, \ldots, 0}^{n-i})$$

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

- Accepting rule

$$Accept \leftarrow Config(s_{acc}, X_1, \ldots, X_n, Y_1, \ldots, Y_n)$$

Datalog[∃,=,⊥]: Overview



But…

– What about joins in rule bodies?

$$\exists E\ Employee(E,D,P,A) \leftarrow Runs(D,P), Area(P,A)$$

– What about the DL assertion concept product?

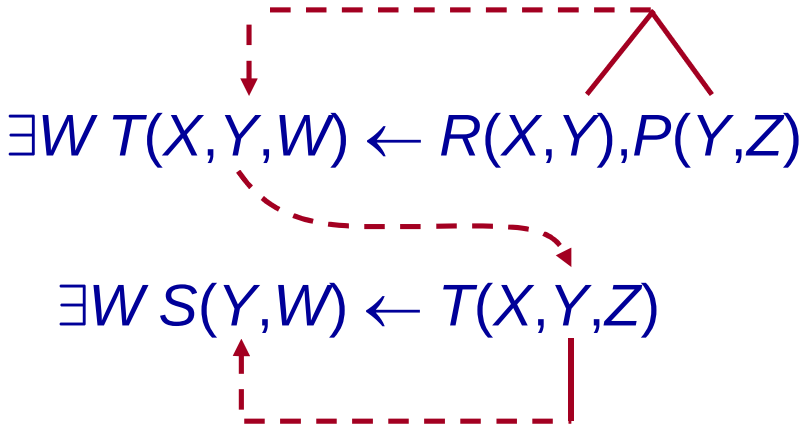
$$biggerThan(E,M) \leftarrow Elephant(E), Mouse(M)$$

Sticky Datalog[∃]

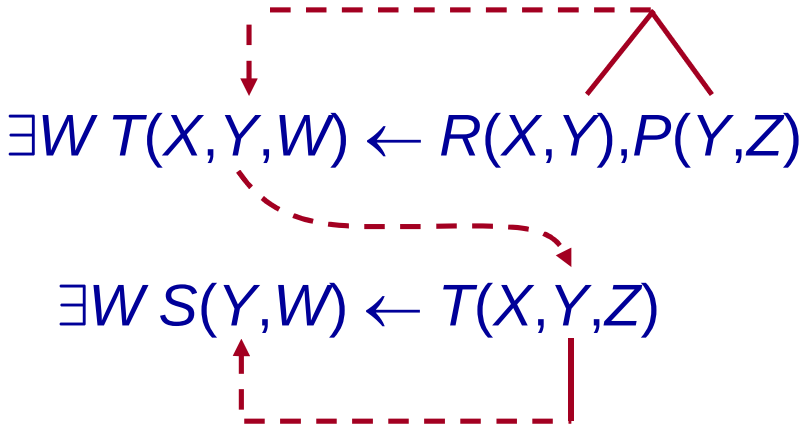
$\exists E\ Employee(E,D,P,A) \leftarrow Runs(D,P), Area(P,A)$

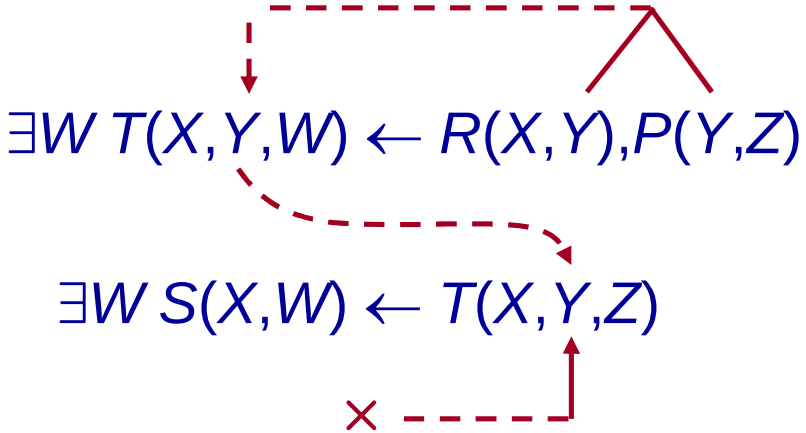
$biggerThan(E,M) \leftarrow Elephant(E), Mouse(M)$

Sticky Datalog[∃]



Sticky Datalog[∃]



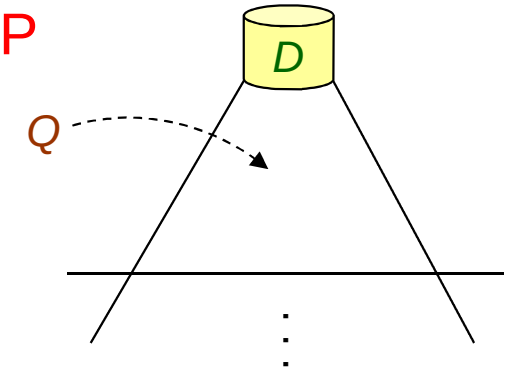


Data Complexity of Sticky Datalog[∃]

Theorem: Sticky Datalog[∃] is in AC_0 in data complexity

Proof:

– Sticky Datalog[∃] enjoys the BDDP



– Thus, Sticky Datalog[∃] is first-order rewritable $\Rightarrow$ in AC_0

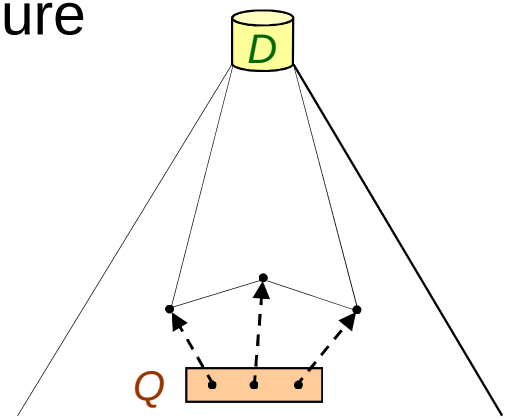
[Calì, Gottlob & Pieris, VLDB 2010]

Combined Complexity of Sticky Datalog[∃]

Theorem: Sticky Datalog[∃] is EXPTIME-complete in combined complexity

Proof:

– EXPTIME-membership: construct a proof of the query by applying an APSPACE procedure



– EXPTIME-hardness: fact inference for lossless Datalog programs over {0,1} is EXPTIME-hard

$S(X,Y,Z) \leftarrow P(X,Y),P(Y,Z),R(Z)$

[Calì, Gottlob & Pieris, VLDB 2010]

Datalog as an Ontology Language

- Ontology languages are based on description logics (previous lecture)
- Much is not possible with Datalog

  [Patel-Schneider & Horrocks, Journal of Web Semantics 2007]

DL Axiom	?
$Employee \sqsubseteq \exists reportsTo$	$\exists Y\ reportsTo(X,Y) \leftarrow Employee(X)$
funct(*reportsTo*)	$Y = Z \leftarrow reportsTo(X,Y), reportsTo(X,Z)$
Employee **disj** *Customer*	$\bot \leftarrow Employee(X), Customer(X)$

Equality Atom

$$X_i = X_j \leftarrow R_1(\mathbf{X_1}),\ldots,R_n(\mathbf{X_n})$$

– Linear Datalog[∃,=] is already undecidable

implicit in [Chandra & Vardi, SIAM Journal on Computing 1985]

– Separability: given $P = P_{\exists} \cup P_{=}$,

$$\forall D \forall Q\colon\ D \models P_{=} \quad \Rightarrow \quad P(D) \models Q \quad \text{iff} \quad P_{\exists}(D) \models Q$$

[Calì, Lembo & Rosati, PODS 2003]

– Non-conflicting Datalog[∃,=]: sufficient condition for separability

see, e.g., [Calì, Gottlob & Lukasiewicz, Journal of Web Semantics 2012]

Truth Constant False

$$\bot \leftarrow R_1(\mathbf{X_1}),\ldots,R_n(\mathbf{X_n})$$

– Preliminary check without adding complexity - given $P = P_\exists \cup P_\bot$

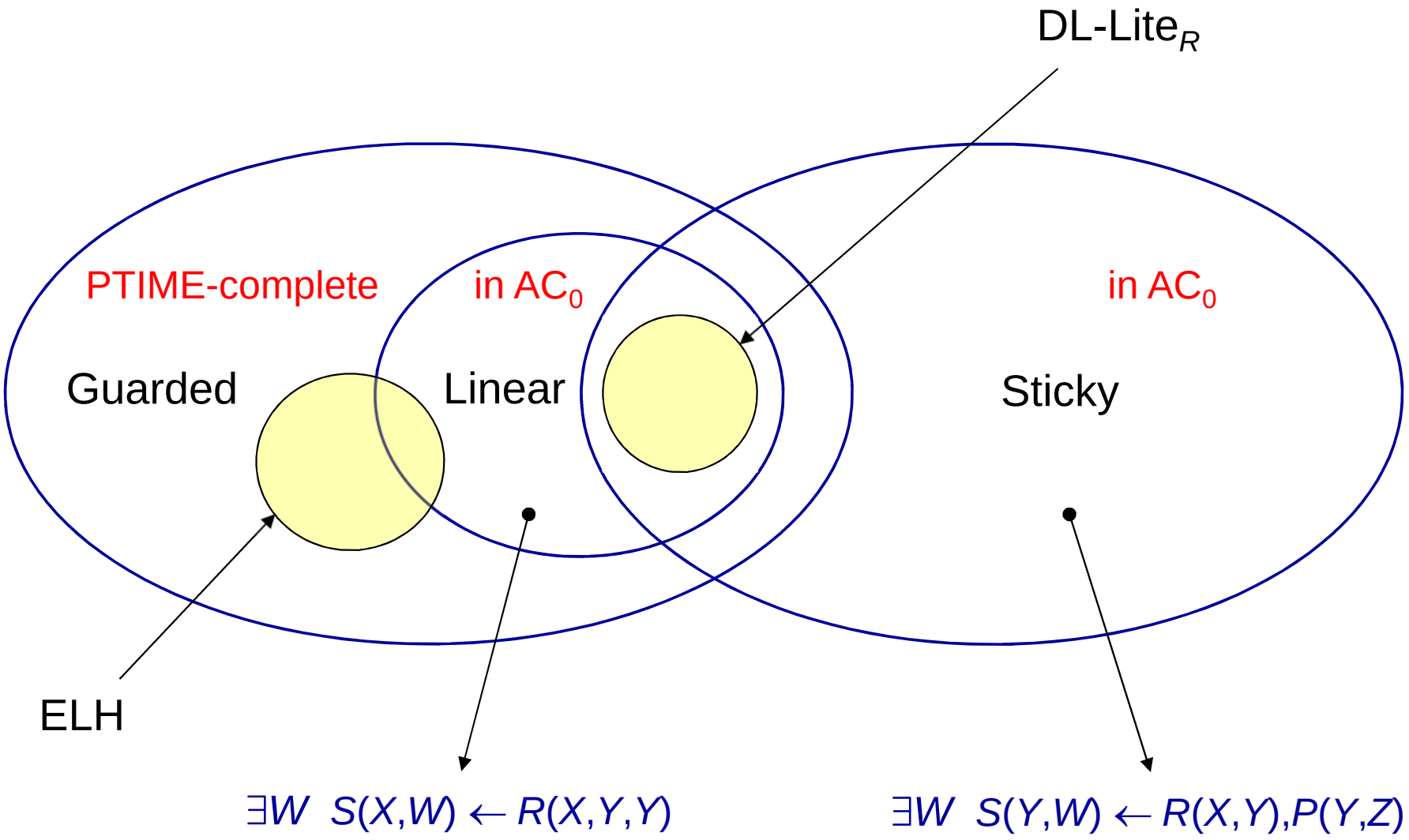
$$P(D) \models Q$$

$$\Updownarrow$$

$P_\exists(D) \models Q$ OR $P_\exists(D) \models Q_\rho$, for some $\rho \in P_\bot$

$$Q_\rho \leftarrow body(\rho)$$

Non-Conflicting Datalog[∃,=,⊥]: Overview



Further Reading (Partial List)

Guarded Datalog[∃] (and extensions)

- Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proceedings of KR*, pages 70-80, 2008.
- Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57-83, 2012.
- Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the complexity lines for generalized guarded existential rules. In *Proceedings of IJCAI*, pages 712-717, 2011.

Sticky Datalog[∃] (and extensions)

- Andrea Calì, Georg Gottlob, Andreas Pieris. Advanced processing for ontological queries. *PVLDB* 3(1): 554-565, 2010.
- Andrea Calì, Georg Gottlob, Andreas Pieris. Query answering under non-guarded rules in Datalog+/-. In *Proceedings of RR*, pages 1-17, 2010.
- Georg Gottlob, Giorgio Orsi, Andreas Pieris. Ontological queries: Rewriting and optimization. In *Proceedings of ICDE*, pages 2-13, 2011.

Weakly-acyclic Datalog[∃] (and extensions)

- Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, Lucian Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1): 89-124, 2005.
- Alin Deutsch, Alan Nash, Jeffrey B. Remmel. The chase revisited. In *Proceedings of PODS*, pages 149-158, 2008.
- Bruno Marnette. Generalized schema-mappings: From termination to tractability. In *Proceedings of PODS*, pages 13-22, 2009.

Shy Datalog[∃]

- Nicola Leone, Marco Manna, Giorgio Terracina, Pierfrancesco Veltri. Efficiently computable Datalog$^{\exists}$ Programs. In *Proceedings of KR*, 2012.