Integración de Bases de Conocimiento

Clase 4 – Datalog+/-.

Profesores: Maria Vanina Martinez y Ricardo Rodriguez

Datalog

- Diseñado para bases de datos *deductivas*:
  - Bases de datos que permiten obtener información que está contenida *implícitamente*.
  - Dos partes: la parte *extensional* y la parte *intensional*; la extensional es un conjunto de *hechos* (proposiciones), la intensional un conjunto de *reglas* que permiten obtener nueva información a partir de la parte extensional.
- Datalog es un lenguaje de programación en lógica (sintácticamente es un subconjunto de *Prolog*).
- Reglas de la forma: $\forall \mathbf{X} \forall \mathbf{Y}\ \phi(\mathbf{X},\mathbf{Y}) \rightarrow R(\mathbf{X})$

Datalog: Poder expresivo

- *No puede expresar* algunos axiomas ontológicos importantes:
  - Inclusión de conceptos que involucran *restricciones existenciales* en roles en la cabeza de las reglas:

$$cientifico \sqsubseteq \exists esAutorde$$

  - Conceptos *disjuntos*:

$$artRevista \sqsubseteq \neg artConferencia$$

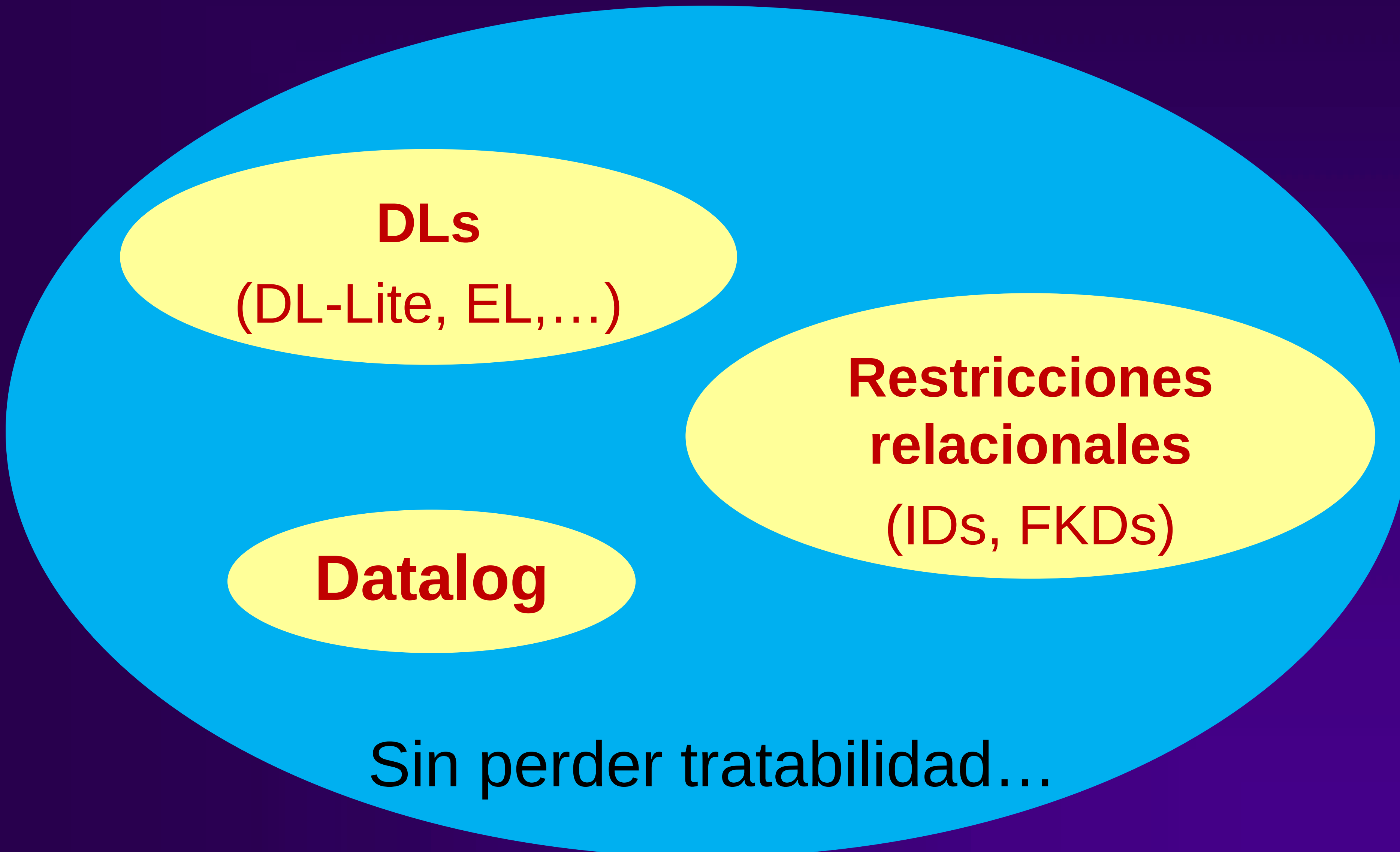
  - *Funciones*: $(funct\ tienePrimerAutor)$
- Buena noticia: ¡Podemos extender Datalog para representar conocimiento ontológico rico!

Razonamiento ontológico y Datalog

DL Assertion	Datalog Rule
Concept Inclusion $emp \sqsubseteq person$	$emp(X) \rightarrow person(X)$
Concept Product $sen\text{-}emp \times emp \sqsubseteq moreThan$	$sen\text{-}emp(X), emp(Y) \rightarrow moreThan(X,Y)$
(Inverse) Role Inclusion $reports^{-} \sqsubseteq mgr$	$reports(X,Y) \rightarrow mgr(Y,X)$
Role Transitivity trans(mgr)	$mgr(X,Y), mgr(Y,Z) \rightarrow mgr(X,Z)$
Participation $emp \sqsubseteq \exists report$	$emp(X) \rightarrow \exists Y\ report(X,Y)$
Disjointness $emp \sqcap customer \sqsubseteq \bot$	$emp(X), customer(X) \rightarrow \bot$
Functionality funct($reports$)	$reports(X,Y), reports(X,Z) \rightarrow Y = Z$

Datalog+/–

Formalismos basados en Datalog



Extendiendo Datalog

- Extensión de Datalog permitiendo *existenciales* en la cabeza de las reglas: $\forall \mathbf{X} \forall \mathbf{Y}\ \Phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\ \Psi(\mathbf{X},\mathbf{Z})$ (TGDs)
- Responder consultas (conjuntivas) en Datalog$^{\exists}$ (extensión con TGDs) *es indecidible* (se puede simular una MT).
- Datalog+/– extiende Datalog con *dependencias* y restricciones de *integridad*… pero con *limitaciones sintácticas* sobre las reglas.
- Datalog+/– es una *familia* de lenguajes ontológicos
  - las distintas restricciones dan lugar a *diferentes lenguajes* con distinto poder expresivo y complejidad computacional (para tareas como *query answering*).

Datalog+/–

- Asumimos:
  - Un universo infinito de constantes Δ
  - Un conjunto infinito de valores nulos (etiquetados) Δ_N
  - Un conjunto infinito de variables $\mathcal{V}$
  - Un esquema relacional $\mathcal{R}$, un conjunto finito de nombres de relaciones (o símbolos predicativos).
- Diferentes constantes representan diferentes valores; diferentes nulls pueden representar el mismo valor.
- Usamos $\mathbf{X}$ para denotar la secuencia $X_1, \ldots, X_n$, $n \geq 0$.
- Una (instancia de) base de datos D sobre $\mathcal{R}$ es un conjunto de átomos con predicados en $\mathcal{R}$ y argumentos en Δ.

Desvío temporario: Homomorfismos

Sean A y B dos *estructuras relacionales finitas* con signatura σ (consistiendo de funciones y relaciones).

- Un *homomorfismo* de A en B es una función h: $dom(A) \rightarrow dom(B)$ que preserva la estructura:
  - para cada función n-aria f en σ y elementos $a_1, ..., a_n \in dom(A)$, vale que $h(f^A(a_1, ..., a_n)) = f^B(h(a_1), ..., h(a_n))$; y
  - para cada relación n-aria R en σ y elementos $a_1, ..., a_n \in dom(A)$, vale que si $(a_1, ..., a_m) \in R^A$, entonces $(h(a_1), ..., h(a_m)) \in R^B$
- Es una relajación del concepto de *isomorfismo* (todo isomorfismo es homomorfismo, pero no al revés).
- La *composición* de homomorfismos es un homomorfismo.

Desvío temporario: Homomorfismos

Sean I y J dos *instancias de BD* sobre un esquema S.

- Para BDs, un *homomorfismo* h: $adom(I) \rightarrow adom(J)$ es una función tal que para cada símbolo relacional $P \in S$ y cada tupla $(a_1, \ldots, a_m)$ tenemos que:

  si $(a_1, \ldots, a_m) \in P^I$, entonces $(h(a_1), \ldots, h(a_m)) \in P^J$

- En BDs *no hay funciones*; la primera condición en la definición anterior no es necesaria (se cumple trivialmente).
- Dos instancias de BD I y J son *homomórficamente equivalentes* si existe un homomorfismo $I \rightarrow J$ y otro $J \rightarrow I$.

Desvío temporario: Homomorfismos

- Lema: Sea Q una BCQ y J una instancia de base de datos. La siguientes afirmaciones son *equivalentes*:
  - $J \models Q$
  - Existe un homomorfismo h: $I^Q \longrightarrow J$.
- Intuitivamente, h corresponde a la *asignación de variables* en Q que hace que ésta se satisfaga en J.

I^Q denota la "*instancia de BD canónica*" de Q, la cual es simplemente un conjunto de hechos construidos a partir de los predicados y variables de Q.

Desvío temporario: Homomorfismos

El "*Problema del homomorfismo*" pregunta simplemente si existe un homomorfismo entre dos estructuras finitas (en este caso, instancias de BD).

- Es *NP-completo*.
- Es un *problema fundamental* para la investigación algorítmica:
  - Todo problema de satisfacción de restricciones es un caso particular; por ejemplo, SAT.
  - Muchos problemas de IA son casos particulares; por ejemplo, planning.

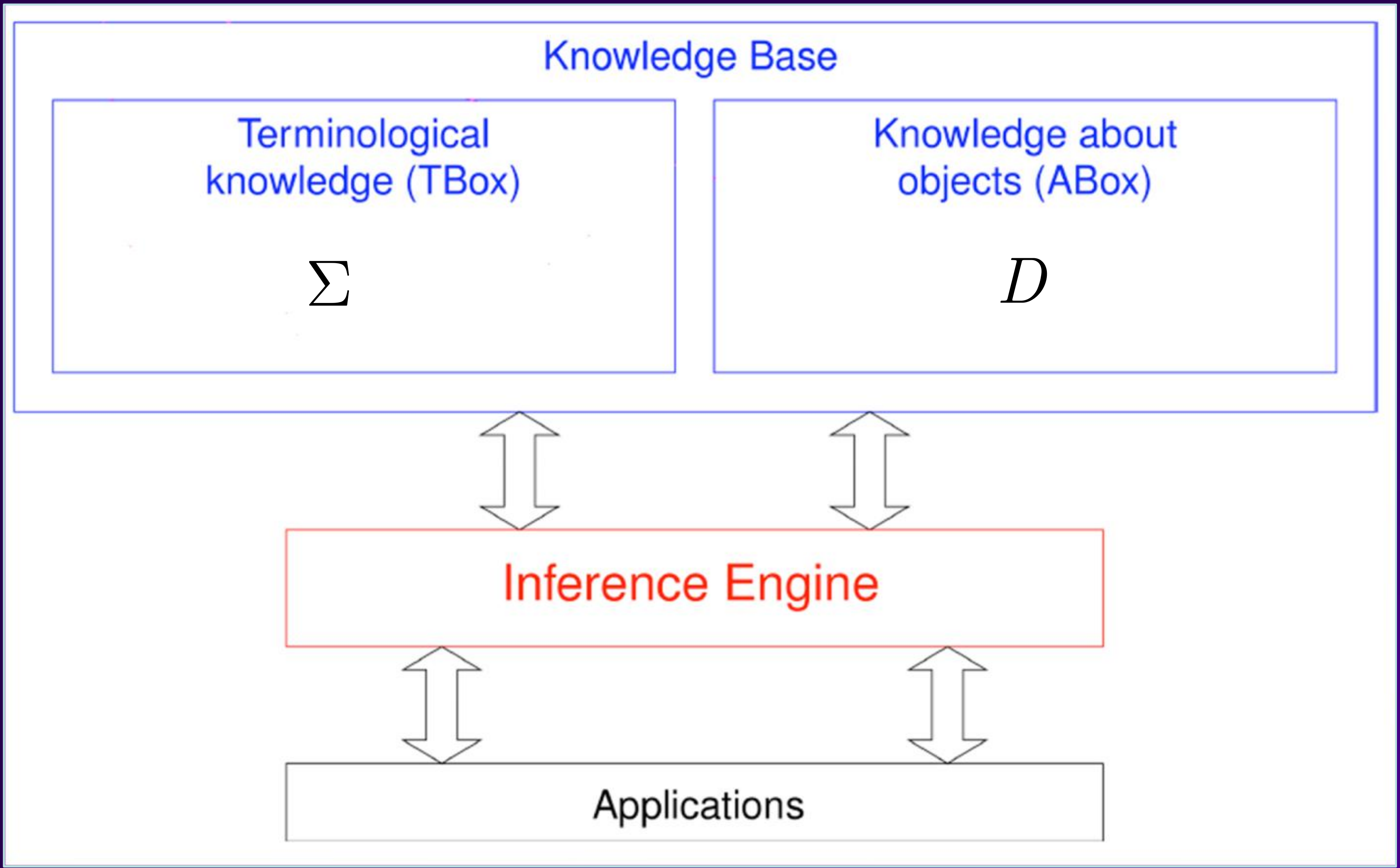
Datalog+/–

- Una consulta conjuntiva (CQ) sobre $\mathcal{R}$ tiene la forma $Q(\mathbf{X}) = \exists \mathbf{Y}\ \Phi(\mathbf{X}, \mathbf{Y})$, Φ es una conjunción de átomos.
- Una consulta conjuntiva Booleana (BCQ) sobre $\mathcal{R}$ tiene la forma $Q() = \exists \mathbf{Y}\ \Phi(\mathbf{X}, \mathbf{Y})$, Φ es una conjunción de átomos.
- Las *respuestas* a una consulta se definen vía homomorfismos, mapeos μ: $\Delta \cup \Delta_N \cup \mathcal{V} \rightarrow \Delta \cup \Delta_N \cup \mathcal{V}$:
  - si $c \in \Delta$ entonces $\mu(c) = c$
  - si $c \in \Delta_N$ entonces $\mu(c) \in \Delta \cup \Delta_N$
  - μ se extiende a (conjuntos de) átomos y conjunciones.
- Conjunto de *respuestas* $Q(D)$: conjunto de tuplas t sobre Δ t.q. $\exists\ \mu$: $\mathbf{X} \cup \mathbf{Y} \rightarrow \Delta \cup \Delta_N$ t.q. $\mu\big(\Phi(\mathbf{X}, \mathbf{Y})\big) \subseteq D$, y $\mu(\mathbf{X}) = t$.

Datalog+/–

- Tuple-generating Dependencies (TGDs) son restricciones de la forma σ: $\forall \mathbf{X} \forall \mathbf{Y}\ \Phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\ \Psi(\mathbf{X},\mathbf{Z})$ donde Φ y Ψ son conjunciones atómicas sobre $\mathcal{R}$:
  - $\forall \mathbf{X} \forall \mathbf{Y}\ \Phi(\mathbf{X},\mathbf{Y})$ se denomina el cuerpo de σ ($body(\sigma)$)
  - $\exists \mathbf{Z}\ \Psi(\mathbf{X},\mathbf{Z})$ se denomina la cabeza de σ ($head(\sigma)$)
- Dada una BD D y un conjunto Σ de TGDs, el conjunto de modelos $mods(D, \Sigma)$ es el conjunto de todos los B tal que:
  - $D \subseteq B$
  - cada $\sigma \in \Sigma$ es satisfecho en B (clásicamente).
- El conjunto de respuestas para una CQ Q en D y Σ, $ans(Q,D,\Sigma)$, es el conjunto de todas las tuplas a tal que $a \in Q(B)$ para todo $B \in mods(D, \Sigma)$.

Arquitectura de un sistema OBDA



Chase

- El Chase es un procedimiento para reparar una BD en relación a un conjunto de dependencias (TGDs).

- (*Informalmente*) Regla de aplicación de TGD:
  - una TGD σ es aplicable a una BD D si $body(\sigma)$ mapea a átomos en D
  - la aplicación de σ sobre D agrega (si ya no existe) un átomo con nulos "frescos" correspondientes a cada una de las variables existenciales cuantificadas en $head(\sigma)$.

Chase

Input: Base de datos D, conjunto de TGDs Y

Output: Un modelo de $D \cup$ Y

D

Person(john)

Σ

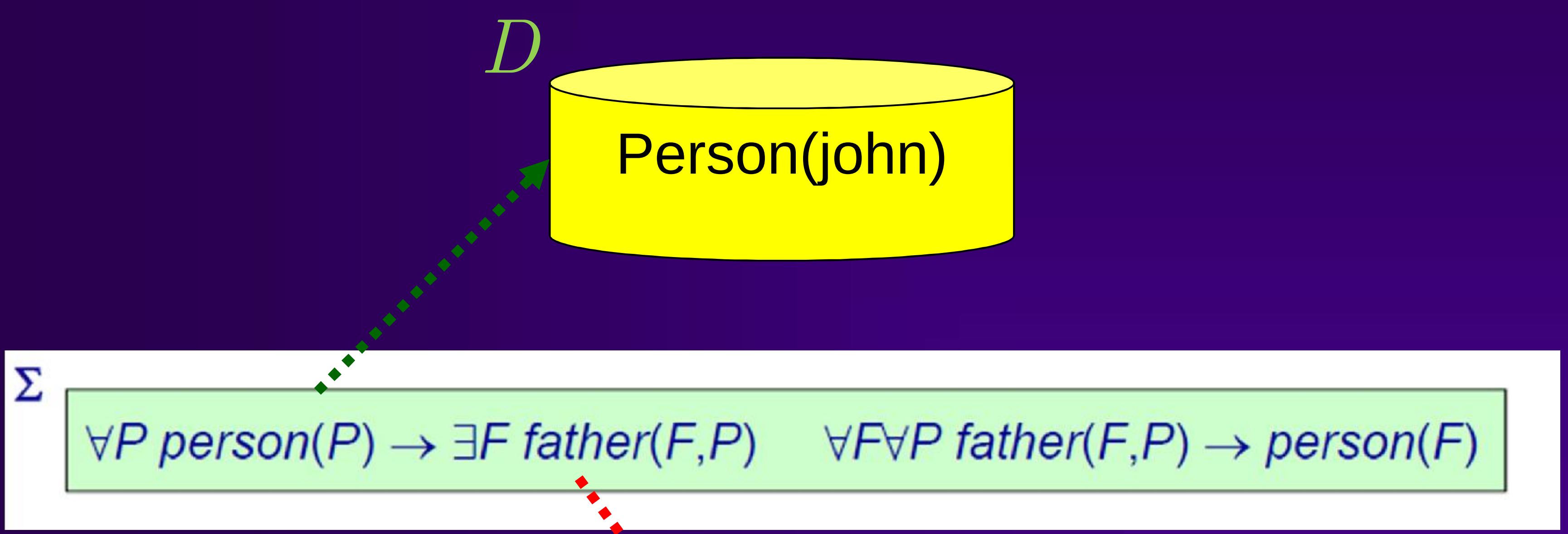
$\forall P\ person(P) \rightarrow \exists F\ father(F,P) \qquad \forall F \forall P\ father(F,P) \rightarrow person(F)$

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

Chase

Input: Base de datos D, conjunto de TGDs Y

Output: Un modelo de $D \cup Y$



$chase(D,Y) = D \cup \{father(z_1, john)\}$

Chase

Input: Base de datos D, conjunto de TGDs Y

Output: Un modelo de $D \cup Y$

D

Person(john)

Σ

$\forall P\ person(P) \rightarrow \exists F\ father(F,P) \quad \forall F \forall P\ father(F,P) \rightarrow person(F)$

$chase(D,Y) = D \cup \{father(z_1,john),\ person(z_1)$

Chase

Input: Base de datos D, conjunto de TGDs Y

Output: Un modelo de $D \cup \mathrm{Y}$

D

Person(john)

Σ

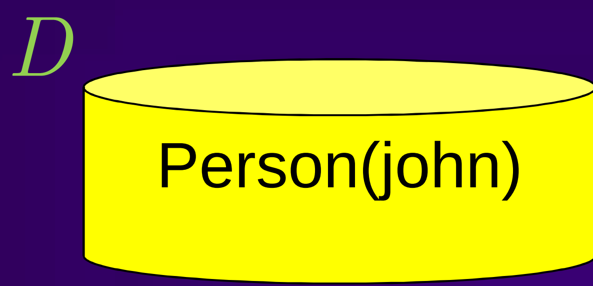
$\forall P\ person(P) \rightarrow \exists F\ father(F,P) \qquad \forall F \forall P\ father(F,P) \rightarrow person(F)$

$$chase(D,\mathrm{Y}) = D \cup \{father(z_1,john),\ person(z_1),\ father(z_2,z_1)\}$$

Chase

Input: Base de datos D, conjunto de TGDs Y

Output: Un modelo de $D \cup$ Y



Σ

$\forall P\ person(P) \rightarrow \exists F\ father(F,P) \qquad \forall F \forall P\ father(F,P) \rightarrow person(F)$

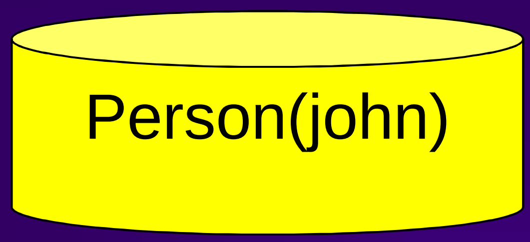
$chase(D,\mathrm{Y}) = D \cup \{father(z_1,john),\ person(z_1),\ father(z_2,z_1),\ldots\}$

Chase

Input: Base de datos D, conjunto de TGDs Y

Output: Un modelo de $D \cup \mathrm{Y}$

D

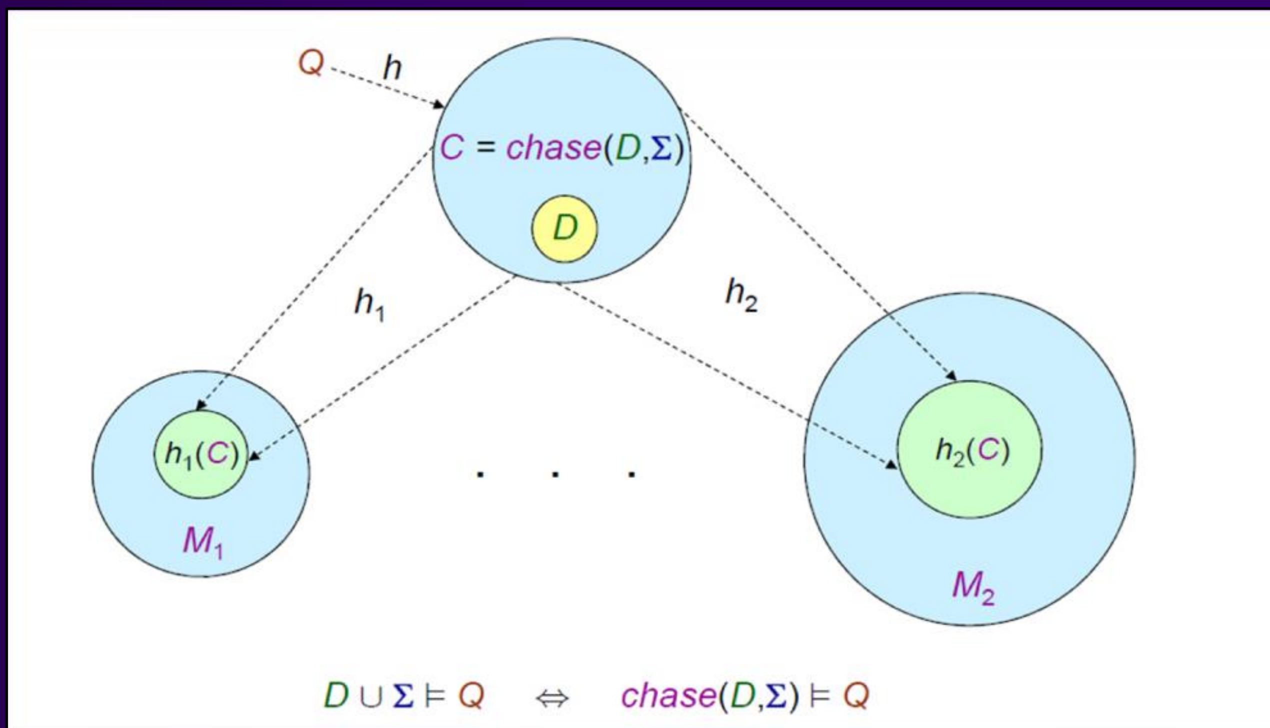


$chase(D,\mathrm{Y}) = D \cup \{father(z_1,john),\ person(z_1),\ father(z_2,z_1), \ldots\}$

INSTANCIA INFINITA

Query Answering vía el chase

- El chase (posiblemente infinito) es un *modelo universal*: existe un homomorfismo de $chase(D, \Upsilon)$ en cada $B \in mods(D, \Upsilon)$.
- Por lo tanto, tenemos que $D \cup \Upsilon \vDash Q$ ssi $chase(D, \Upsilon) \vDash Q$.



Negative Constraints y EGDs

- *Negative constraints* (NCs) son fórmulas de la forma $\forall \mathbf{X}\ \Phi(\mathbf{X}) \rightarrow \perp$, donde $\Phi(\mathbf{X})$ es a conjunción of átomos.
- Las NCs son fáciles de verificar: podemos verificar que la CQ $\Phi(\mathbf{X})$ tiene un conjunto vacío de respuestas en D y Υ.
- *Equality Generating Dependencies* (EGDs) son de la forma $\forall \mathbf{X}\ \Phi(\mathbf{X}) \rightarrow X_i = X_j$, donde Φ es una conjunción of átomos y X_i , X_j son variables que aparecen en $\mathbf{X}$.
- Se asume un conjunto de EGDs separables; intuitivamente significa que las EGDs y TGDs son independientes entre sí.

Datalog+/–: Ejemplo

$D = \{\ directs(john,\ sales),\ directs(anna,\ sales),$
$directs(john,\ finance),\ supervises(anna,john),$
$works\_in(john,sales),\ works\_in(anna,sales)\}$

$\Sigma_T = \{works\_in\ (\mathrm{X},D) \rightarrow\ emp(X),$

$manager(X) \rightarrow\ \exists Y\ supervises(X,Y),$

$supervises(X,Y) \wedge directs(X,D) \rightarrow works\_in(Y,D)\}$

$\Sigma_{NC} = \{supervises(X,Y) \wedge manager(Y) \rightarrow \perp,$

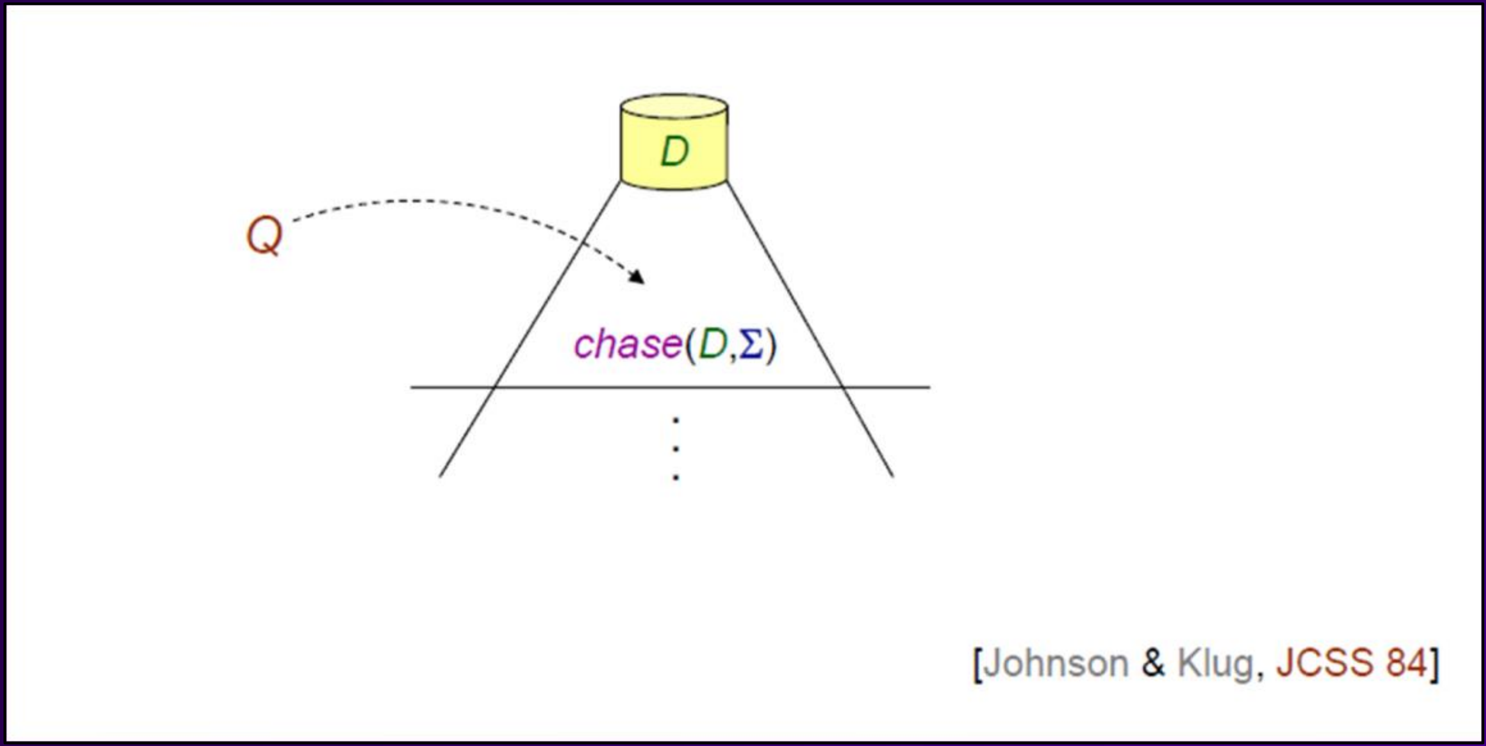
$supervises(X,Y) \wedge works\_in(X,D) \wedge directs(Y,D) \rightarrow \perp,$

$directs(X,D) \wedge directs\ (X,D') \rightarrow D = D'\}$

Resultados positivos

Query Answering con IDs es decidible

- PSPACE-completo en complejidad *combinada*
- NP-completo complejidad *ba-combinada*



Guarded Datalog+/–

- Una TGD se dice guarded si existe un átomo en su cuerpo que contiene todas las variables que aparecen en el cuerpo.

$$\forall \mathrm{X} \forall \mathrm{Y} \forall \mathrm{Z}\ \underbrace{R(\mathrm{X},\mathrm{Y},\mathrm{Z})}_{\color{red}{guard}},\ S(\mathrm{Y}),\ P(\mathrm{X},\mathrm{Z}) \rightarrow \exists \mathrm{W}\ \ Q(\mathrm{X},\mathrm{W})$$

- El *chase* tiene *treewidth finito* $\Rightarrow$ query answering decidible
- Query answering es PTIME-completo en complejidad *data*.
- Extiende la Lógica de descripción ELH (misma complejidad data).

Guarded Datalog+/–

- ELH lógica de descripción muy popular para representar datasets biológicos con complejidad data PTIME.

EL TBox	Datalog$^{\pm}$ Representation
$A \sqsubseteq B$	$\forall X\, A(X) \rightarrow B(X)$
$A \sqcap B \sqsubseteq C$	$\forall X\, A(X), B(X) \rightarrow C(X)$
$\exists R.A \sqsubseteq B$	$\forall X\, R(X,Y), A(Y) \rightarrow B(X)$
$A \sqsubseteq \exists R.B$	$\forall X\, A(X) \rightarrow \exists Y\, R(X,Y), B(Y)$
$R \sqsubseteq P$	$\forall X \forall Y\, R(X,Y) \rightarrow P(X,Y)$

Linear Datalog+/–

- Una TGD se dice *linear* (lineal) si tiene sólo un átomo en su cuerpo.

$$\forall \mathbf{X} \forall \mathbf{Y}\ R(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\ Q(\mathbf{X},\mathbf{Z})$$

$guard$

- Las linear TGDs son (trivialmente) *guarded.*
- Query answering está en AC_0 en complejidad *data* (*reescritura de primer orden – FO rewritablity*).
- Extiende la (familia de) lógicas de descripción *DL-Lite* (misma complejidad data).

Linear Datalog+/–

- DL-Lite familia de lógicas de descripción con data complejidad AC_0 (OWL 2 QL).

DL-Lite TBox	Datalog$^{\pm}$ Representation
$A \sqsubseteq B$	$\forall X\, A(X) \rightarrow B(X)$
$A \sqsubseteq \exists R$	$\forall X\, A(X) \rightarrow \exists Y\, R(X,Y)$
$\exists R \sqsubseteq A$	$\forall X \forall Y\, R(X,Y) \rightarrow A(X)$
$R \sqsubseteq P$	$\forall X \forall Y\, R(X,Y) \rightarrow P(X,Y)$

TGDs FO Re-escribibles

Q

Σ

TGDs FO Re-escribibles

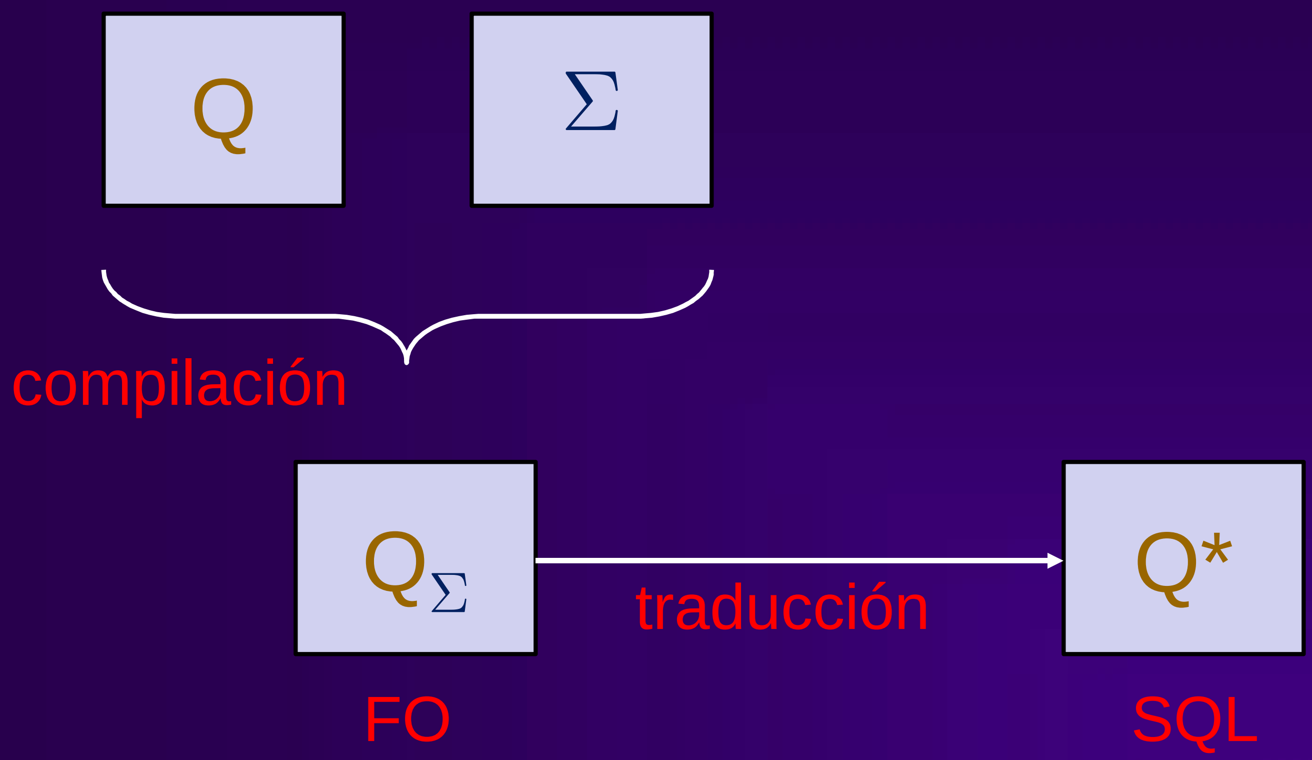
Q Σ

compilación

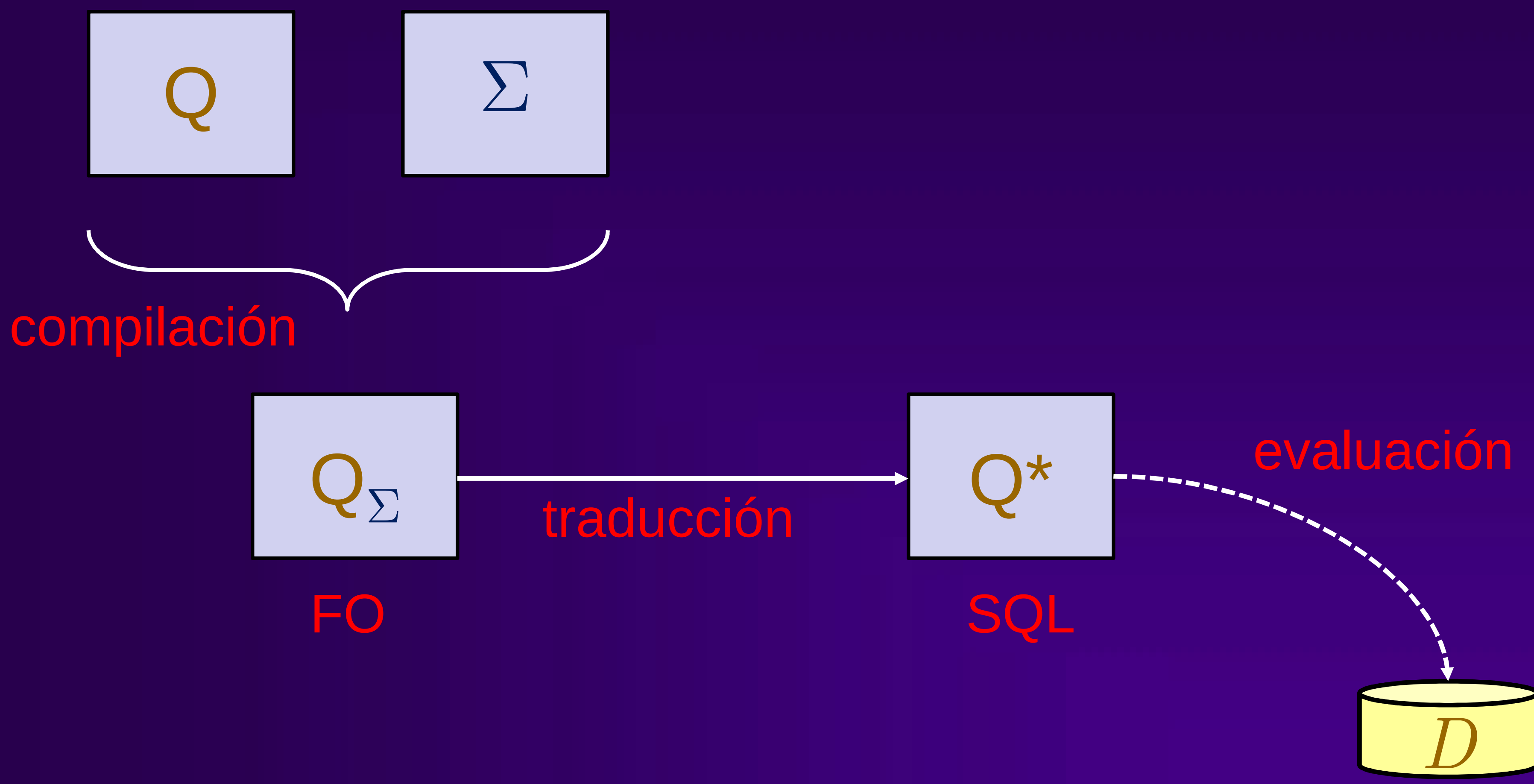
Q_{Σ}

FO

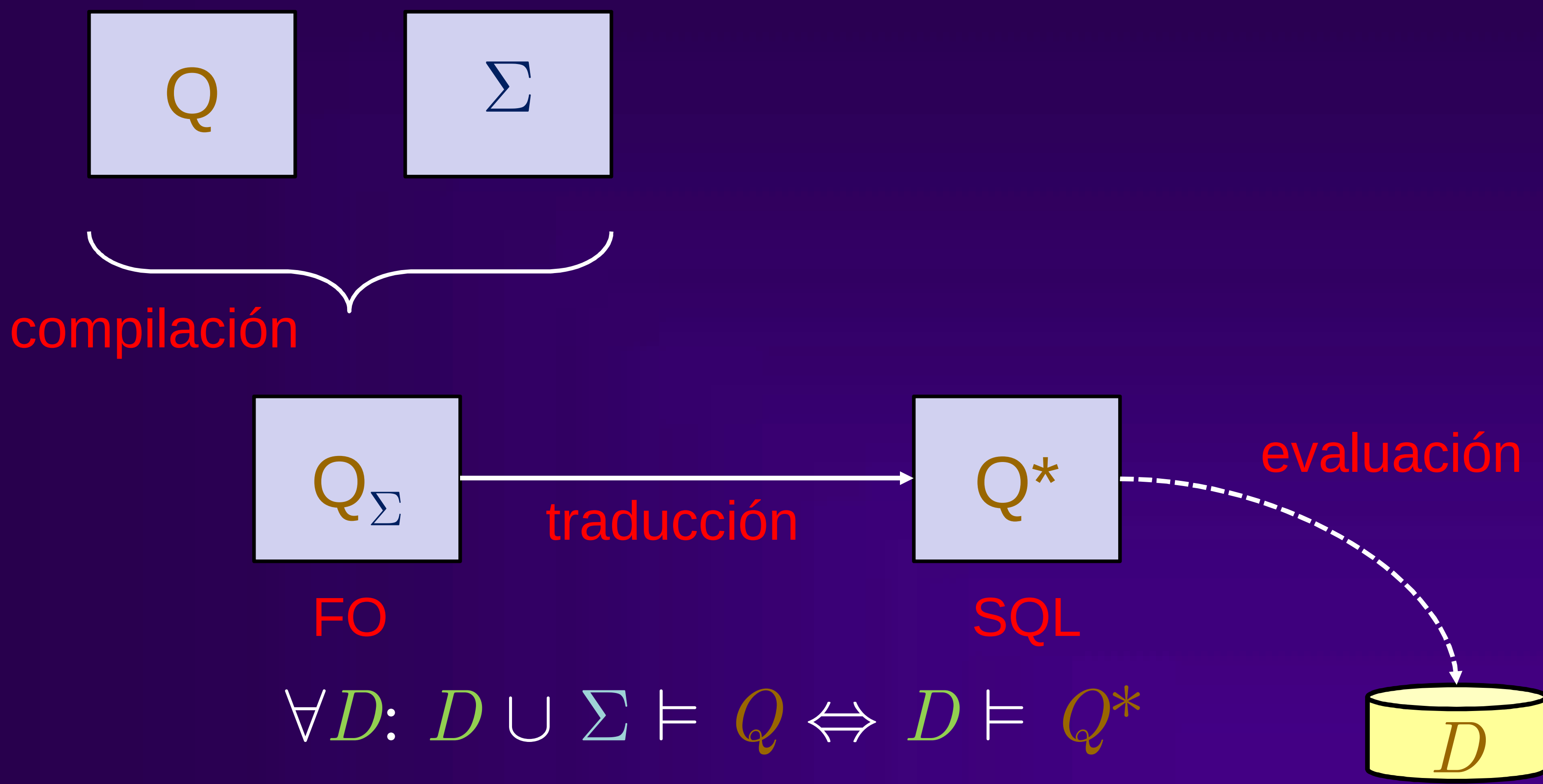
TGDs FO Re-escribibles



TGDs FO Re-escribibles



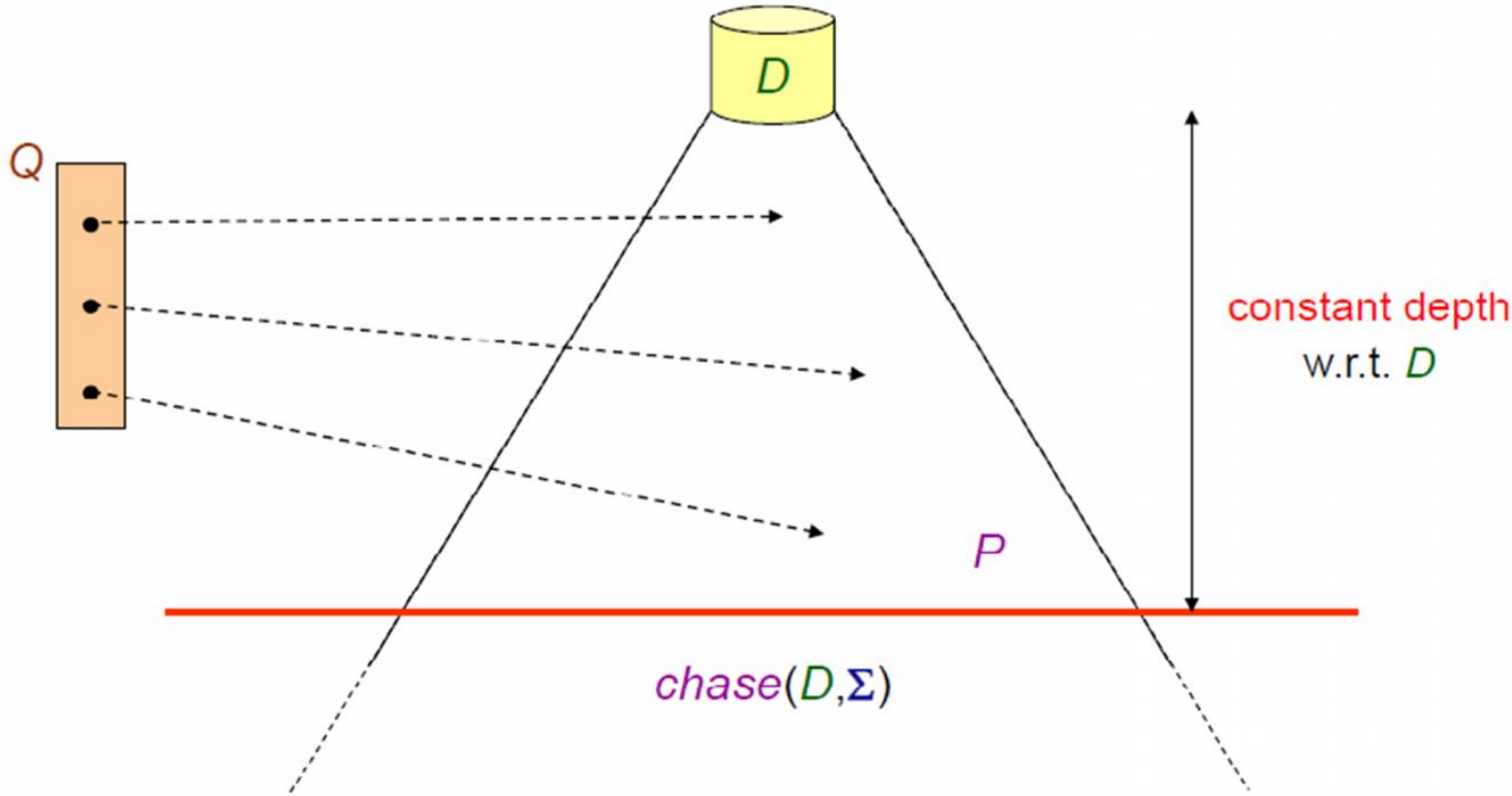
TGDs FO Re-escribibles



$$\forall D\colon\ D \cup \Sigma \models Q \Leftrightarrow D \models Q^*$$

Query answering está en AC_0
(complejidad data)

Bounded Derivation-Depth Property (BDDP)



$chase(D,\Sigma) \models Q \quad \Rightarrow \quad P \models Q$

Bounded Derivation-Depth Property (BDDP)

- Query answering en programas Datalog$^{\exists}$ que satisfacen BDDP está en AC_0 (complejidad data).
- *Bounded derivation-depth* es una propiedad *semántica*.
- Queremos identificar un *fragmento sintáctico* de Datalog$^{\exists}$ que satisfaga la propiedad.
- Algunos *resultados*:
  - Guarded Datalog$^{\exists}$ no satisface BDDP (*¿cómo lo probaría?*)
  - Linear Datalog$^{\exists}$ satisface BDDP.
  - BDDP $\Rightarrow$ FO rewritability

Pero…

- ¿Qué sucede con los *joins* en los cuerpos de las reglas?

$$\forall A\ \forall D\ \forall P\ runs(D,P),\ area(P,A) \rightarrow \exists E\ employee(E,D,P,A)$$

- ¿Y los axiomas de *productos* de conceptos (cartesiano)?

$$\forall E \forall M\ elephant(E),\ mouse(M) \rightarrow biggerThan(E,M)$$

- No se pueden garantizar modelos con *forma de árbol*

$$\left.\begin{array}{l}\forall X \forall Y\ R(X,Y) \rightarrow \exists Z\ R(Y,Z) \\ \forall X \forall Y\ R(X,Y) \rightarrow S(X)\end{array}\right\}$$

Infinita cantidad de símbolos en $\mathbb{S}$

$$\left.\forall X \forall Y\ S(X),\ S(Y) \rightarrow P(X,Y)\right\}$$

P forma un clique infinito

Stickiness

$$\forall A \forall D \forall P \; runs(D,P),\; area(P,A) \rightarrow \exists E \; employee(E,D,P,A)$$

$$\forall E \forall M \; elephant(E), mouse(M) \rightarrow biggerThan(E,M)$$

Stickiness

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

$$\forall X \forall Y \forall Z \ T(X,Y,Z) \rightarrow \exists W \ S(Y,W)$$

✓

Stickiness

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

$$\forall X \forall Y \forall Z\ T(X,Y,Z) \rightarrow \exists W\ S(X,W)$$

Stickiness: procedimiento de marcado

- Marcado Inicial: marcar *todas* las ocurrencias de las *variables del cuerpo* de la regla que no aparecen en *todos* los átomos de la cabeza.

$$\forall V \forall W\, R_1(V,W) \rightarrow \exists X \exists Y \exists Z\, R_2(W,X,Y,Z)$$

$$\forall V \forall W \forall X \forall Y\, R_2(V,W,X,Y) \rightarrow \exists Z\, R_1(W,Z), R_3(W,Y), R_4(Y,X)$$

$$\forall W \forall X \forall Y\, R_3(W,X), R_4(X,Y) \rightarrow \exists Z\, R_5(Z,Y,X)$$

Stickiness: procedimiento de marcado

- Marcado Inicial: marcar *todas* las ocurrencias de las *variables del cuerpo* de la regla que no aparecen en *todos* los átomos de la cabeza.
- *Propagación*: propagar el marcado de las variables del cuerpo vía los átomos de la cabeza.

$$\forall V \forall W\, R_1(V,W) \;\rightarrow\; \exists X \exists Y \exists Z\, R_2(W,X,Y,Z)$$

$$\forall V \forall W \forall X \forall Y\, R_2(V,W,X,Y) \;\rightarrow\; \exists Z\, R_1(W,Z), R_3(W,Y), R_4(Y,X)$$

$$\forall W \forall X \forall Y\, R_3(W,X), R_4(X,Y) \;\rightarrow\; \exists Z\, R_5(Z,Y,X)$$

Stickiness: procedimiento de marcado

- Las variables marcadas ocurren sólo una vez en el cuerpo de cada TGD.

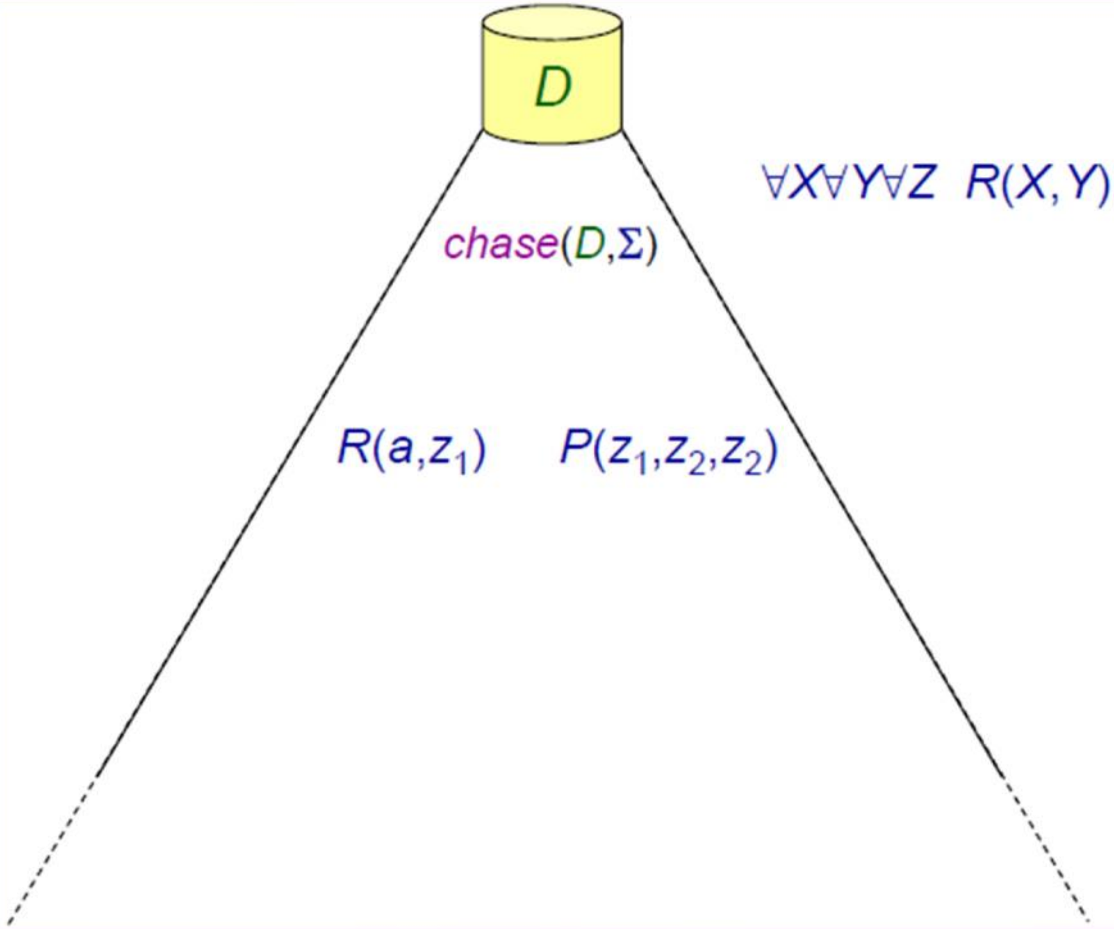
$$\forall V \forall W \; R_1(V,W) \rightarrow \exists X \exists Y \exists Z \; R_2(W,X,Y,Z)$$

$$\forall V \forall W \forall X \forall Y \; R_2(V,W,X,Y) \rightarrow \exists Z \; R_1(W,Z), R_3(W,Y), R_4(Y,X)$$

$$\forall W \forall X \forall Y \; R_3(W,X), R_4(X,Y) \rightarrow \exists Z \; R_5(Z,Y,X)$$

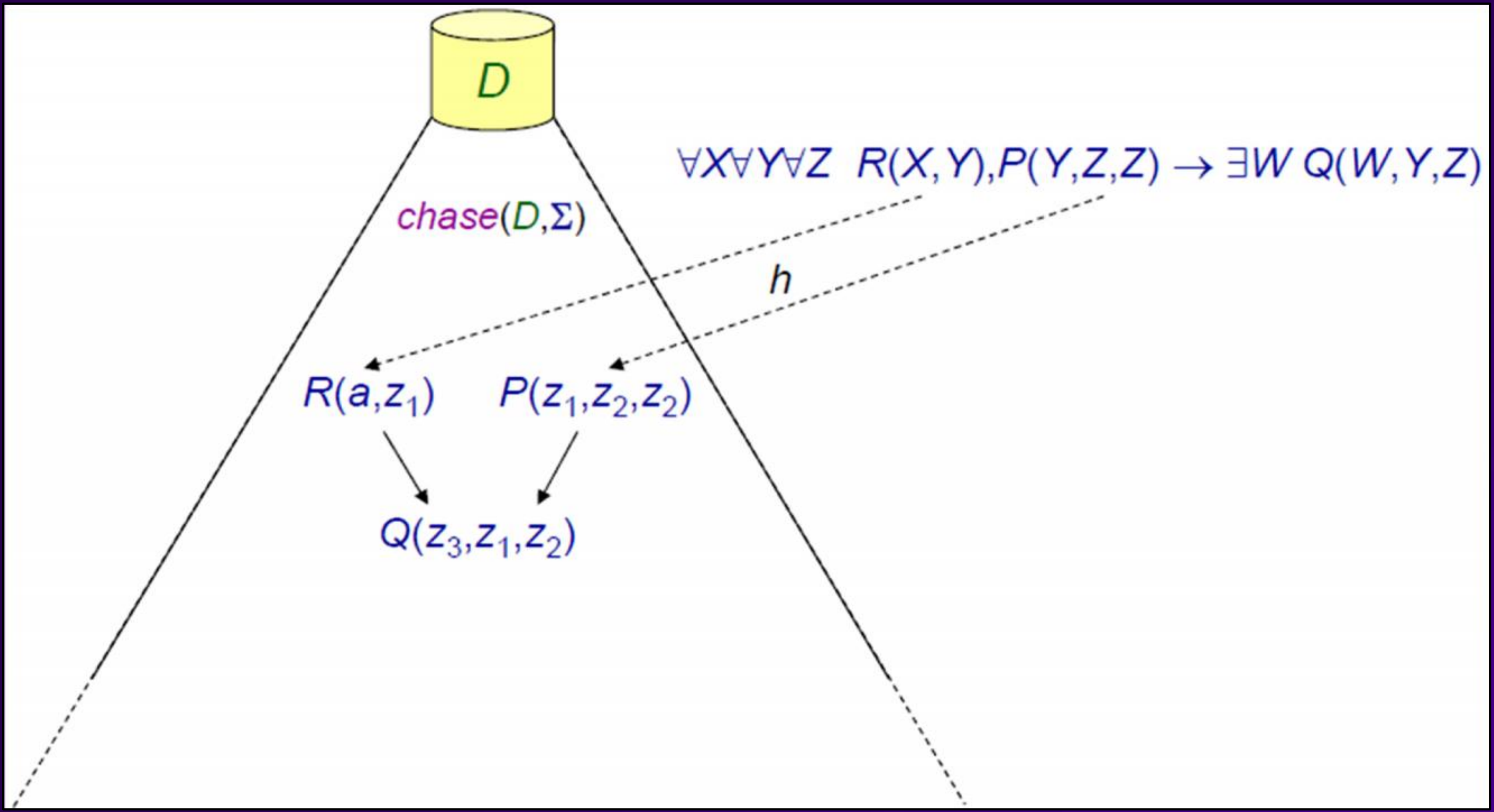
- El *chase* tiene la propiedad *sticky* (backward-resolution termina)

Stickiness

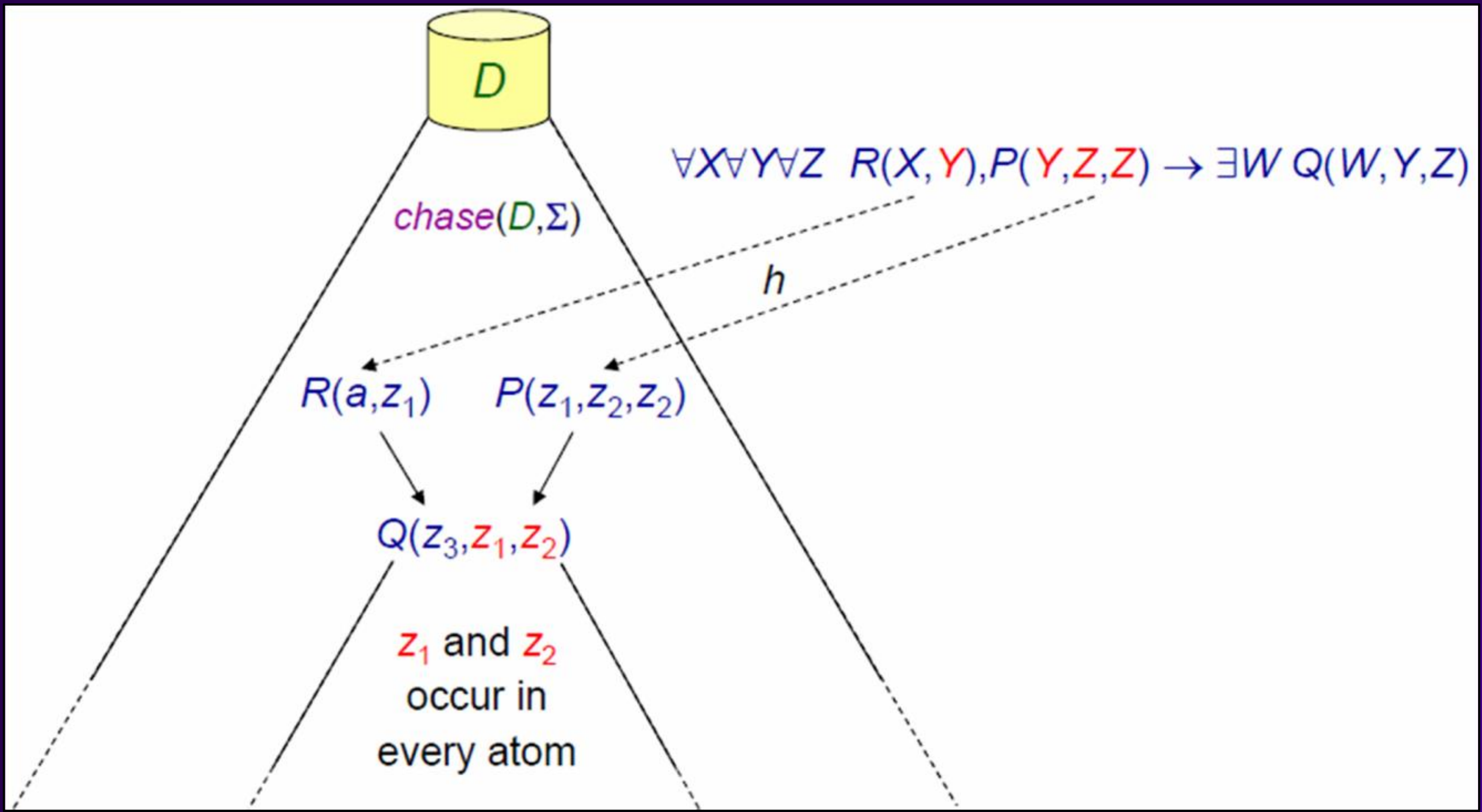


$\forall X \forall Y \forall Z \;\; R(X,Y), P(Y,Z,Z) \rightarrow \exists W \; Q(W,Y,Z)$

Stickiness



Stickiness



D
$\forall X \forall Y \forall Z \;\; R(X,Y), P(Y,Z,Z) \rightarrow \exists W \; Q(W,Y,Z)$
chase(D,Σ)
h
$R(a,z_1)$
$P(z_1,z_2,z_2)$
$Q(z_3,z_1,z_2)$
z_1 and z_2
occur in
every atom

Stickiness: procedimiento de marcado

- Las variables marcadas ocurren sólo una vez en el cuerpo de cada TGD.

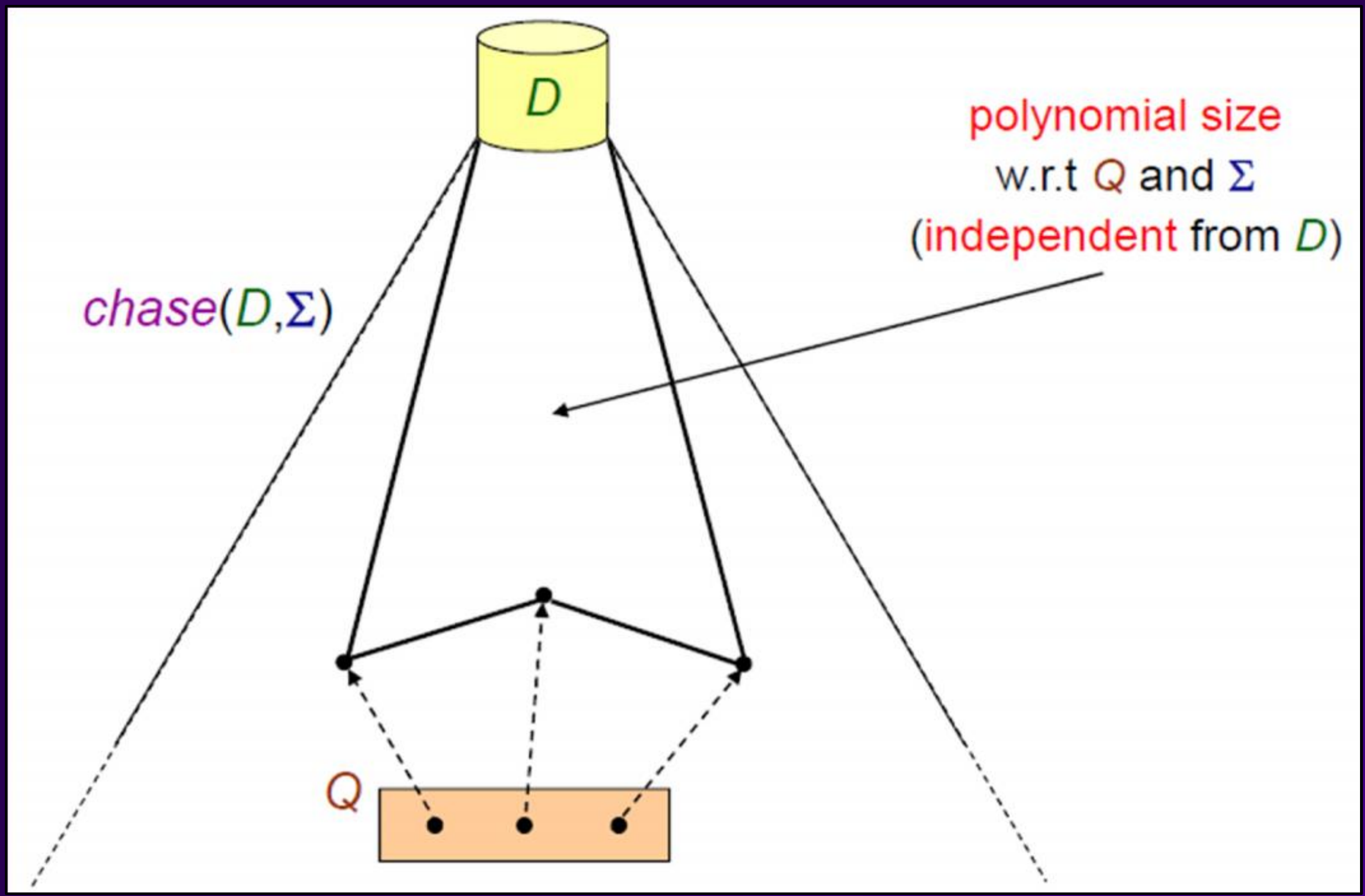
$$\forall V \forall W \;\; R_1(V,W) \;\rightarrow\; \exists X \exists Y \exists Z \; R_2(W,X,Y,Z)$$

$$\forall V \forall W \forall X \forall Y \;\; R_2(V,W,X,Y) \;\rightarrow\; \exists Z \; R_1(W,Z), R_3(W,Y), R_4(Y,X)$$

$$\forall W \forall X \forall Y \;\; R_3(W,X), R_4(X,Y) \;\rightarrow\; \exists Z \; R_5(Z,Y,X)$$

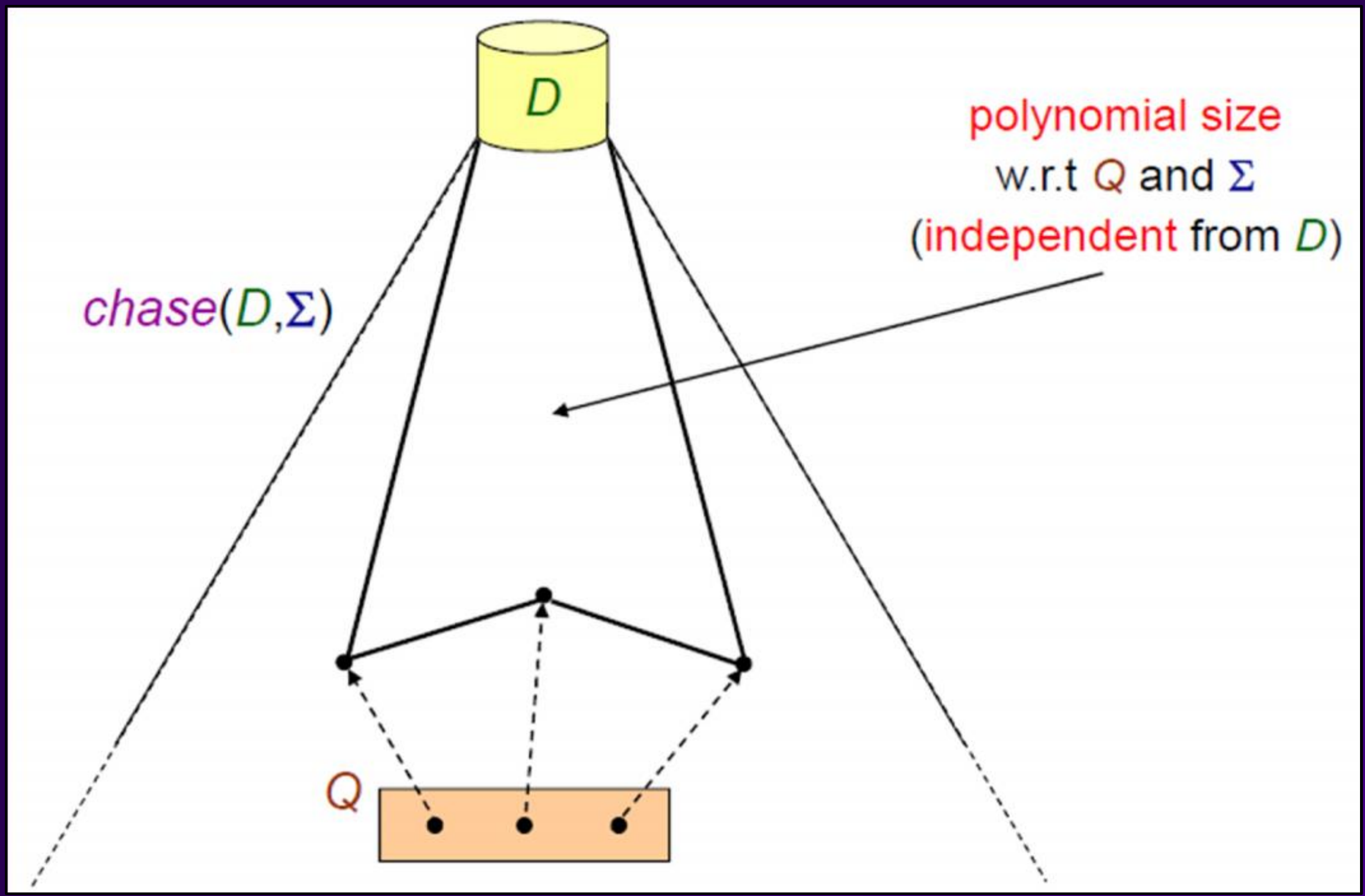
- Query answering en Sticky Datalog+/– está en AC_0 (data complejidad – FO rewritability)
- Extiende a la familia DL-Lite (misma complejidad data)

Polynomial Witness Property (PWP)



- PWP ⇒ re-escritura en *Datalog* (no recursivo) de tamaño polinomial.

Polynomial Witness Property (PWP)



- Linear y Sticky TGDs tienen la propiedad PWP.

Finite Controllability

$$\mathrm{D} \cup \Sigma \models \mathrm{Q} \quad \overset{?}{\Leftrightarrow} \quad \mathrm{D} \cup \Sigma \models_{fin} \mathrm{Q}$$

- La propiedad vale para:
  - *Dependencias de inclusión*
  - Guarded TGDs
  - Sticky TGDs

Otras propiedades

- EGDs: $\forall X\,\forall Y\,\forall Z\ reports(X,Y),\ reports(Y,Z) \rightarrow\ Y = Z$
  - *Non-Conflicting* EGDs: no *interactúan* con el conjunto de TGDs.
  - Chequeo de *satisfabilidad* no agrega complejidad (misma complejidad que *query answering* para el fragmento al que pertenece $D \cup \Sigma$).
- *Negative constraints*: $\forall X\ emp(X),\ customer(X) \rightarrow \bot$
  - Se puede *verificar* si $D \cup \Sigma$ satisface el conjunto de NCs sin agregar complejidad.

EGDs: Finite Controllability

- Finite controllability no vale en general en la presencia de EGDs *arbitrarias*.

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

$$\Sigma = \left\{ \begin{array}{l} \forall X \forall Y\ R(X,Y) \rightarrow \exists Z\ R(Y,Z) \\ \forall X \forall Y \forall Z\ R(Y,X), R(Z,X) \rightarrow Y = Z \end{array} \right\}$$

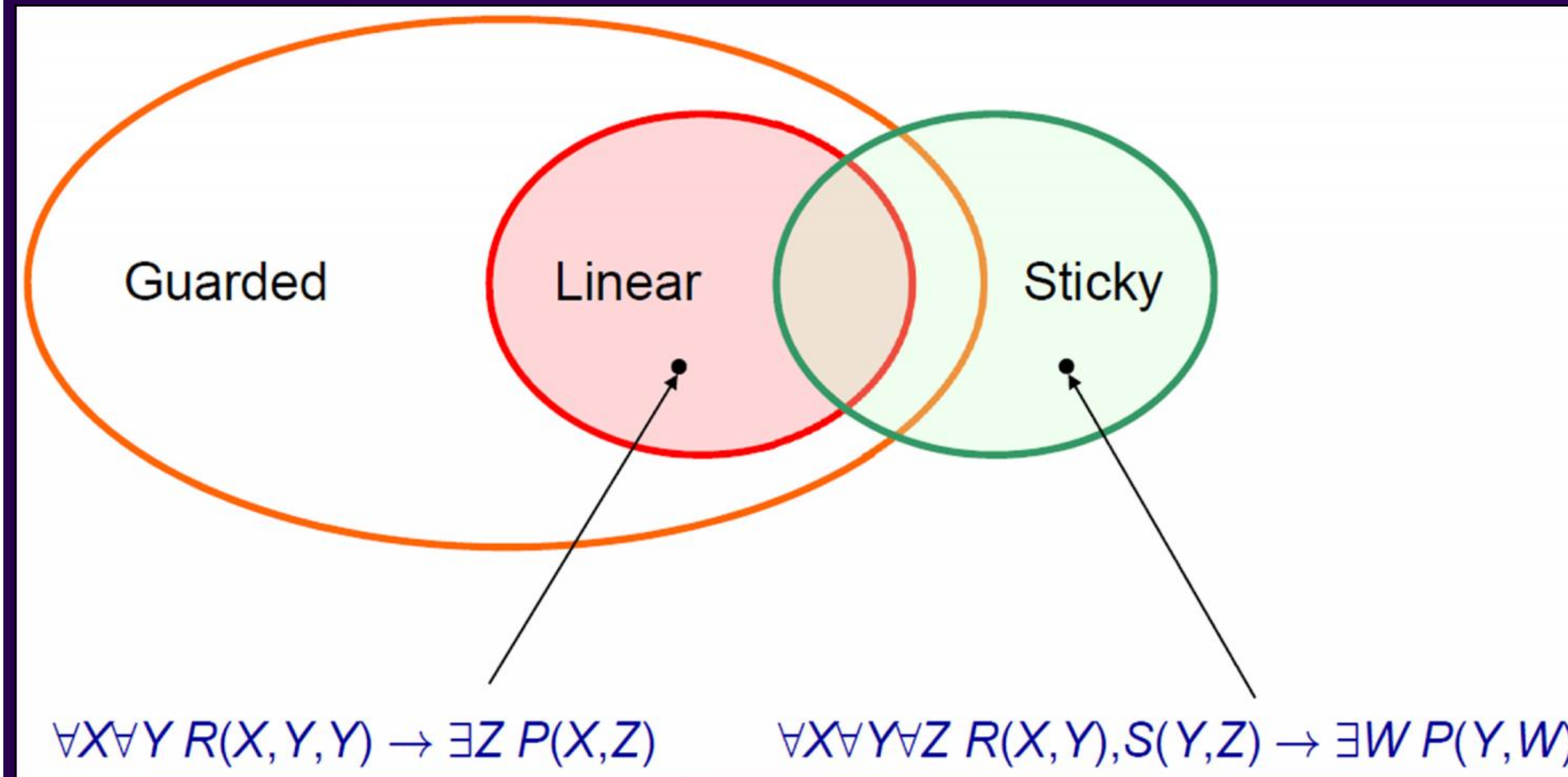
$Q \leftarrow R(A,a)$

$D \cup \Sigma \not\models Q$

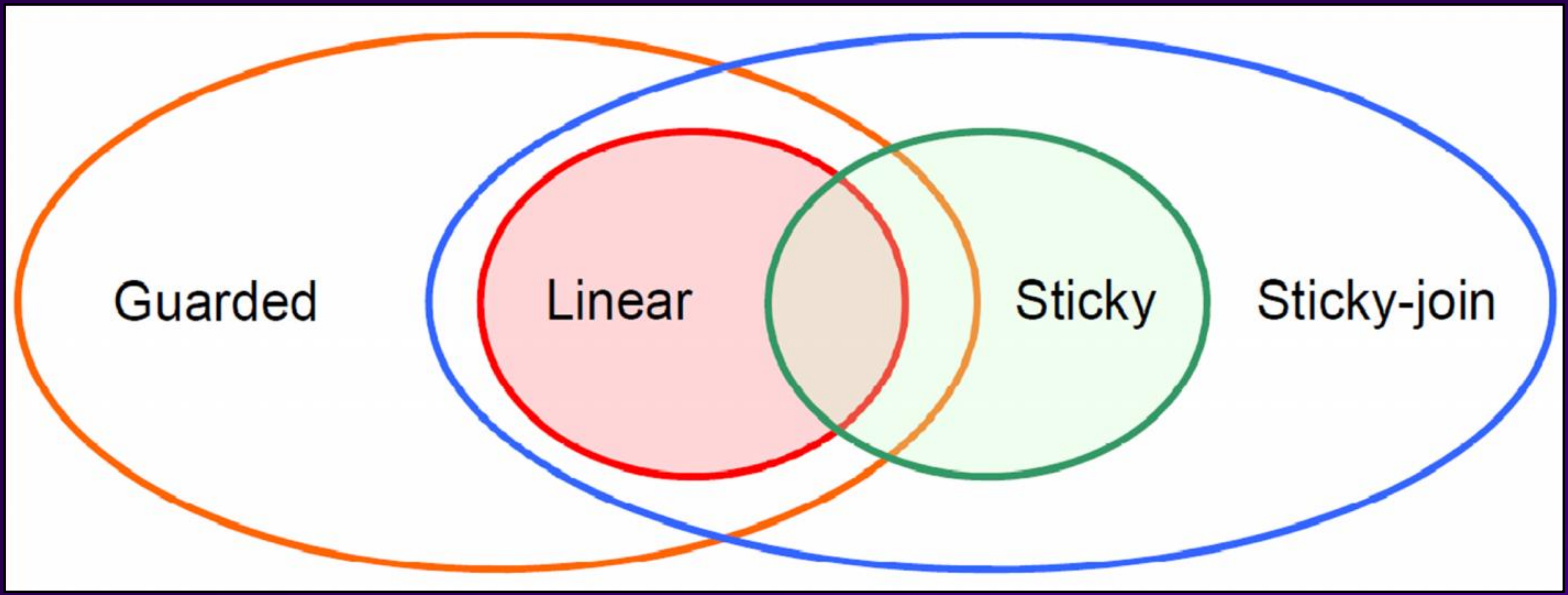
but

$D \cup \Sigma \models_{\text{fin}} Q$

Datalog+/–: Resumen

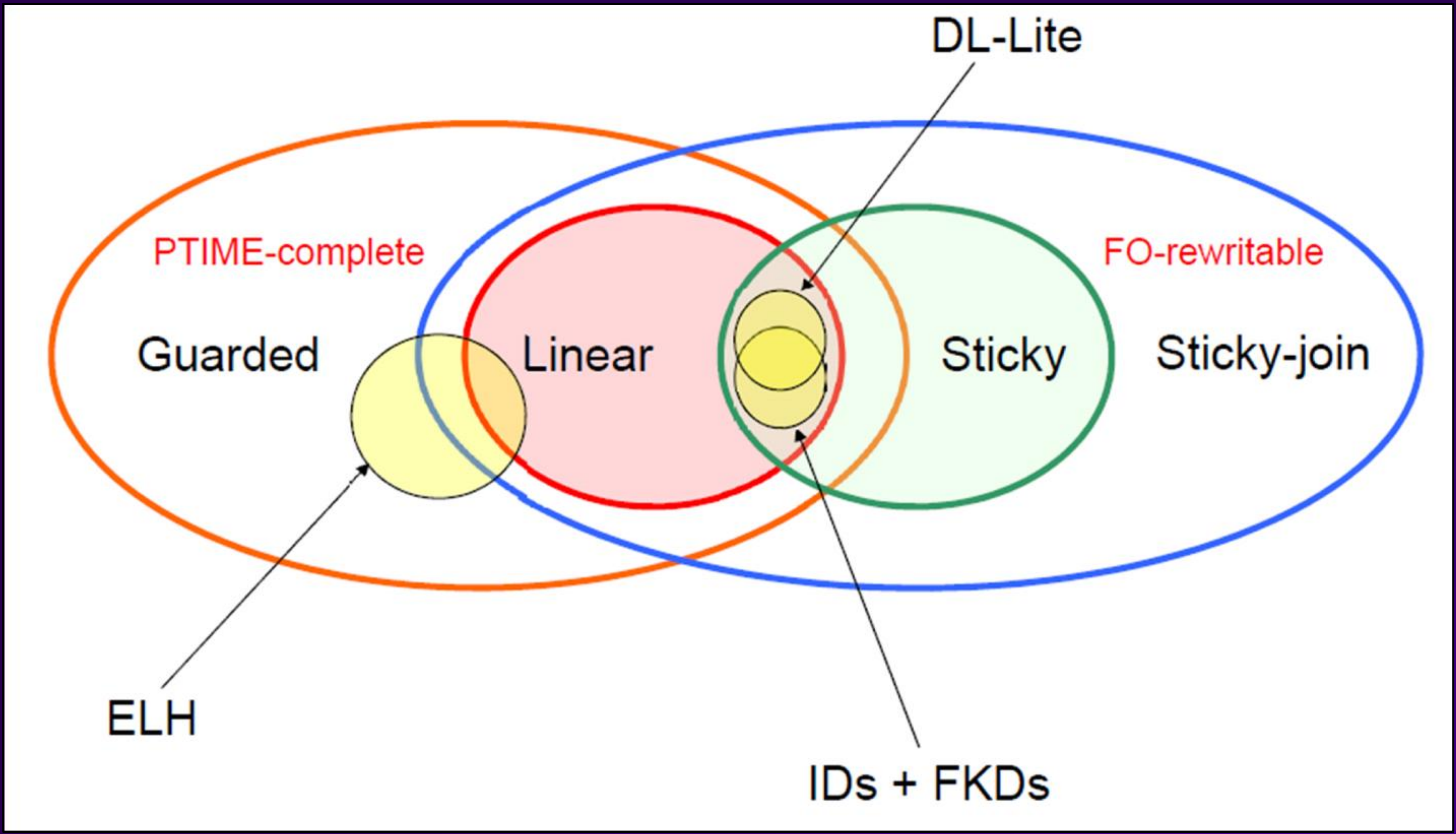


Datalog+/–: Resumen



- Sticky-join: sin perder FO rewritability y Polynomial Witness property (PWP) … pero *más difíciles de identificar*: PSPACE-completo

Datalog+/–: Resumen



Datalog+/–: Resumen

	Data	Fixed Σ	Combined
Guarded	PTIME-complete	NP-complete	2EXPTIME-complete
Linear	in AC_0	NP-complete	PSPACE-complete
Sticky	in AC_0	NP-complete	EXPTIME-complete
Sticky-join	in AC_0	NP-complete	EXPTIME-complete

- Misma complejidad con NCs y EGDs *no conflictivas*.
- Misma complejidad bajo *modelos finitos*.

Clases de complejidad y circuitos

Clase de complejidad $\mathrm{NC_i}$:

- Para $i \geq 0$: Un lenguaje $L \subseteq \{0, 1\}^*$ está en $\mathrm{NC_i}$ si existe una familia de circuitos con fan-in 2 $\{\mathrm{C}_n\}$, una MT determinista M y una constante k tal que:
  - L es *aceptado* por $\{\mathrm{C}_n\}$
  - M *genera* $\{\mathrm{C}_n\}$ y funciona en espacio logarítmico
  - $profundidad(\mathrm{C}_n) \leq k * (log\ n)^i$

Clases de complejidad y circuitos

Clases de complejidad $\mathrm{AC_i}$ y $\mathrm{NC_i}$:

$$\mathrm{NC_0} \subseteq \mathrm{AC_0} \subseteq \mathrm{NC_1} \subseteq \mathrm{AC_1} \subseteq \ldots \subseteq \mathrm{AC_k} \subseteq \mathrm{NC_{k+1}}$$

$$\mathrm{AC_0} \subseteq \mathrm{NC_1} \subseteq \mathrm{LOGSPACE} \subseteq \mathrm{NC_2}$$

$\mathrm{AC_0}$: Circuitos tienen profundidad constante.

$\mathrm{AC_1}$: Circuitos tienen profundidad logarítmica.

Esto nos permite resolver muchos problemas.

- *Si L es un lenguaje regular, entonces $L \in \mathrm{NC_1}$.*
- Sea $\Sigma = \{0,1\}$:
  - $1.\ \Sigma^*\ .1 \in \mathrm{NC_0}$
  - $\{0^k\ 1^k\} \in \mathrm{AC_0}$
  - $\mathrm{PARES} := \{w \in \Sigma^* \mid w \text{ tiene un número par de 1's}\} \in \mathrm{NC_1}$

Consultas FO: Complejidad data

- <u>Teorema</u>: Query answering para consultas FO es *completo* para $\mathrm{logtime\text{-}uniform\ AC_0}$ en complejidad data.

- Prueba:
  - Membresía en $\mathrm{AC_0}$: Dada una base de datos, se *construye* un circuito. La consulta está fija.
  - Hardness: *Transformar* los circuitos a consultas FO (transformación logtime).

Consultas FO: Complejidad data

- El esquema y la consulta se asumen *fijos*.
- La base de datos y el tamaño del dominio activo *pueden variar*.
- Familias *uniformes*: el número total de compuertas de entrada se determina unívocamente por el tamaño del *dominio activo*.
- Ejemplo:
  - Esquema: $R(A,B,C),\ S(D),\ T(E,F)$
  - Número de compuertas de entrada: $n^3 + n + n^2$ para algún n (es el tamaño de dominio activo).

Circuitos Booleanos: Ejemplo

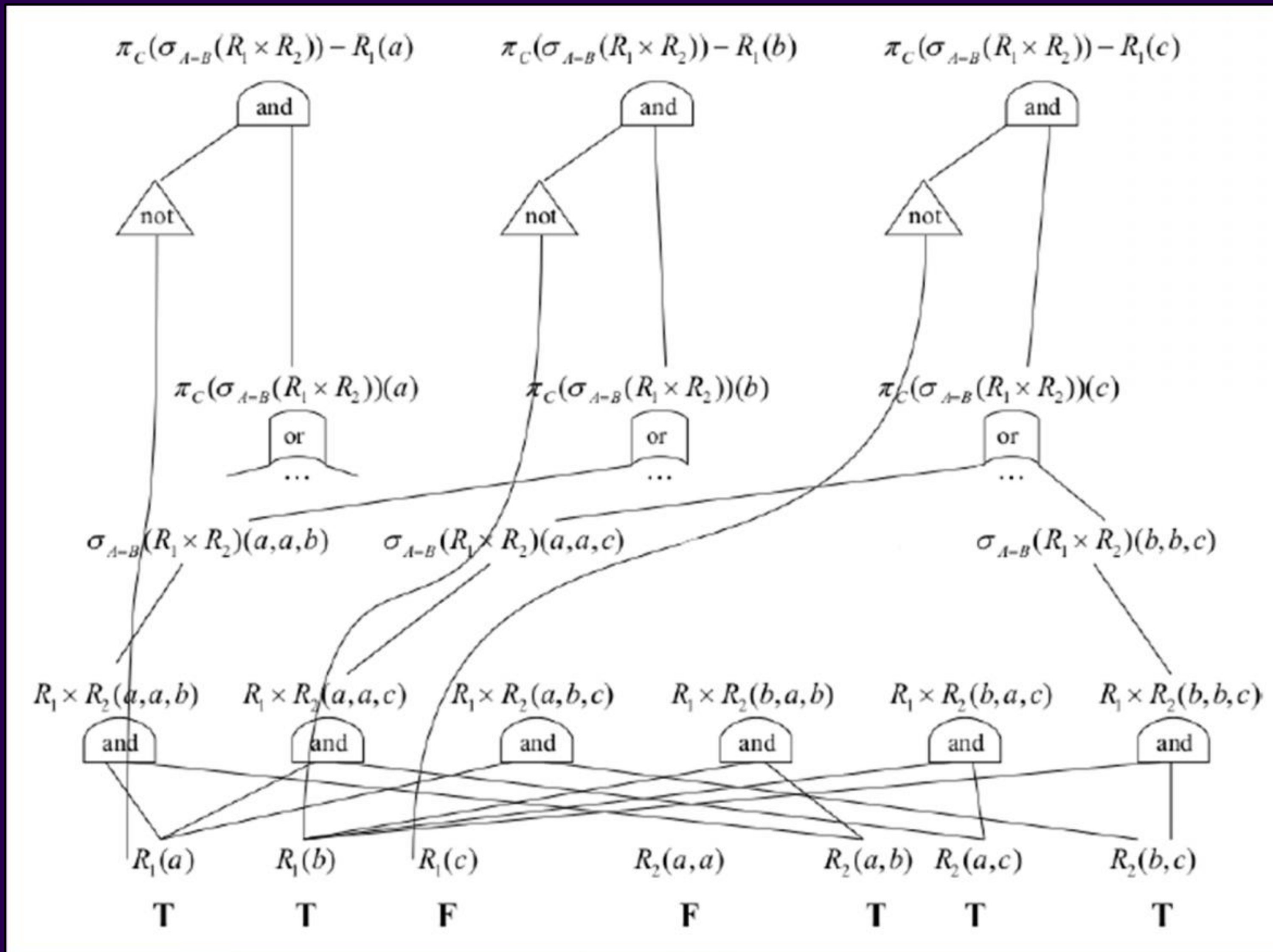
$$\pi_C(\sigma_{A=B}(R_1 \times R_2)) - R_1$$

R_1	A
	a
	b

R_2	B	C
	a	b
	a	c
	b	c

$R_1(a)$	$R_1(b)$	$R_1(c)$	$R_2(a,a)$	$R_2(a,b)$	$R_2(a,c)$	$R_2(b,c)$
T	**T**	**F**	**F**	**T**	**T**	**T**

Circuitos Booleanos: Ejemplo



Referencias

[NB2012] Daniele Nardi and Ronald J. Brachman. 2003. "*An introduction to description logics*". The Description Logic Handbook, Cambridge University Press, New York, NY, USA pp. 1–40.

[CL2007] Diego Calvanese Domenico Lembo. 2007. "*Ontology-based Data Access*". Tutorial at the 6th International Semantic Web Conference (ISWC 2007).

[Johnson & Klug JCSS 84] D.S. Johnson and A. Klug. "*Testing containment of conjunctive queries under functional and inclusion dependencies*". JCSS, 28:167189, 1984.

"Theory of Data and Knowledge Bases", dictado originalmente en TU Wien por Georg Gottlob y luego en University of Oxford por Georg Gottlob y Thomas Lukasiewicz.

M. Arenas: "*Complejidad basada en circuitos*". Complejidad Computacional – IIC3242, Pontificia Universidad Católica de Chile, 2014.

Referencias

Parte del contenido de este curso está basado en:

- *Trabajo de investigación realizado en colaboración con Thomas Lukasiewicz, Georg Gottlob, V.S. Subrahmanian, Avigdor Gal, Andreas Pieris, Giorgio Orsi, Livia Predoiu y Oana Tifrea-Marciuska.*
- Y el siguiente curso: “Methods and Tools for Developing Ontology-Based Data Access Solutions Concepts for Ontology-Based Data Access” dictado por Giuseppe De Giacomo, Domenico Lembo, Antonella Poggi, Valerio Santarelli and Domenico Fabio Savo, en ISWC 2017: https://sites.google.com/a/dis.uniroma1.it/mt4obda/

DLs: otros constructores

- Disyunción: $\forall hasChild.(Doctor \sqcup Lawyer)$
- Restricciones de valores: $\forall tieneHijo.Femenino$
  - Equivalente a $\forall Y\ tieneHijo(X,Y) \rightarrow Femenino(Y)$ en FOL
- Restricciones existenciales: $\exists tieneHijo.Femenino$
  - Equivalente a $\exists Y\ tieneHijo(X,Y) \rightarrow Femenino(Y)$ en FOL.
- Restricciones de número: representan restricciones de cardinalidad en los individuos de los conceptos. Por ejemplo, $(\geq 3\ tieneHijo) \sqcap (\leq 2\ familiaresFemeninos)$
- Negación de conceptos: $\neg(Doctor \sqcup Lawyer)$
- Inverse role: $\forall hasChild^{-}.Doctor$
- Reflexive-transitive role closure: $\exists hasChild^{*}.Doctor$