

Integración de Bases de Conocimiento Datos

2do Cuatrimestre de 2020

Clase 4: Razonamiento (Onto)lógico



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Profesora: Vanina Martinez

mvmartinez@dc.uba.ar

Características de un sistema de SRR

1. Arrancamos definiendo el dominio de la tarea que queremos caracterizar. Este dominio es donde el sistema tendrá que responder consultas, resolver un problema o directamente interactuar: diseño de una edificio, el sistema eléctrico de una casa, la estructura de las clases dictadas en una universidad, etc.
2. Describir las cosas de las cuales queremos “hablar” en el sistema, los individuos y sus relaciones: en el edificio, las oficinas, las conexiones y los empleados pueden ser importantes pero los pizarrones colgados en las paredes pueden no interesarnos. **Definir la ontología del dominio.**



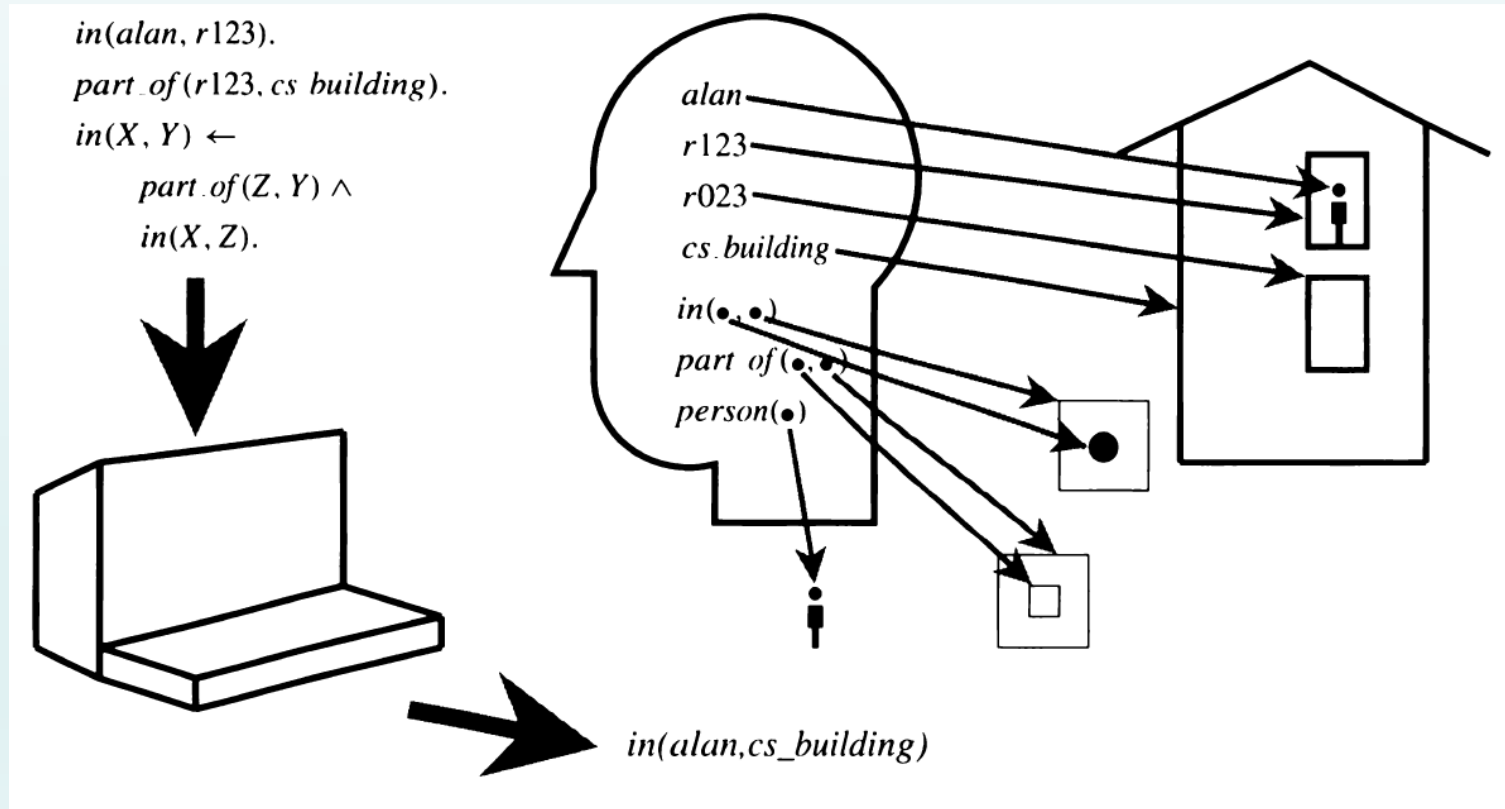
Características de un sistema de SRR

3. Usar símbolos en la computadora para representar los individuos y relaciones. Estos símbolos *denotan* objetos en el mundo real. El símbolo r_{123} podría denotar una oficina particular, y el símbolo *alan* podría representar a una persona particular.
4. Transmitirle a la computadora conocimiento del dominio: tanto conocimiento de lo que es verdadero (o existe) en el dominio como conocimiento sobre cómo resolver el problema: si estamos diseñando un sistema de seguridad que identifique quién está en el edificio deberíamos decirle la disposición de los oficinas, los empleados, la de reconocerlos, etc.



Características de un sistema de SRR

5. Hacerle al sistema una consulta para la cual necesite razonar sobre su conocimiento. Por ejemplo, podemos preguntarle, si *alan* está en edificio.



Representación y Razonamiento

- En un sistema de Representación de Conocimiento y Razonamiento se distinguen los siguientes componentes:
 1. Un **lenguaje formal de representación**.
 2. Una **semántica** que relaciona la representación con su significado.
 3. Una **teoría de raciocinio** o **teoría de prueba** o **procedimiento de prueba** que implementa la máquina de inferencia.



La implementación de un RRS

- En un sistema de Representación de Conocimiento y Razonamiento se distinguen los siguientes componentes:
 1. Un **parser para el lenguaje**: un lenguaje que distingue las sentencias válidas del lenguaje y produce una estructura interna que la represente.
 2. Un **procedimiento de razonamiento**: la implementación de una teoría de prueba y un algoritmo de búsqueda adecuado. Resuelve el no determinismo de las teorías de prueba.

La semántica no se implementa: nos da un marco de cómo el sistema debe comportarse. El procedimiento puede ser o no correcto (sano y completo) con respecto a la semántica.



Lógica vrs Control

Un SRR permite separar ambas cosas:

- Definir la **lógica del dominio**.
- La especificación del **control** para encontrar una solución adecuada.

Esta independencia hace que:

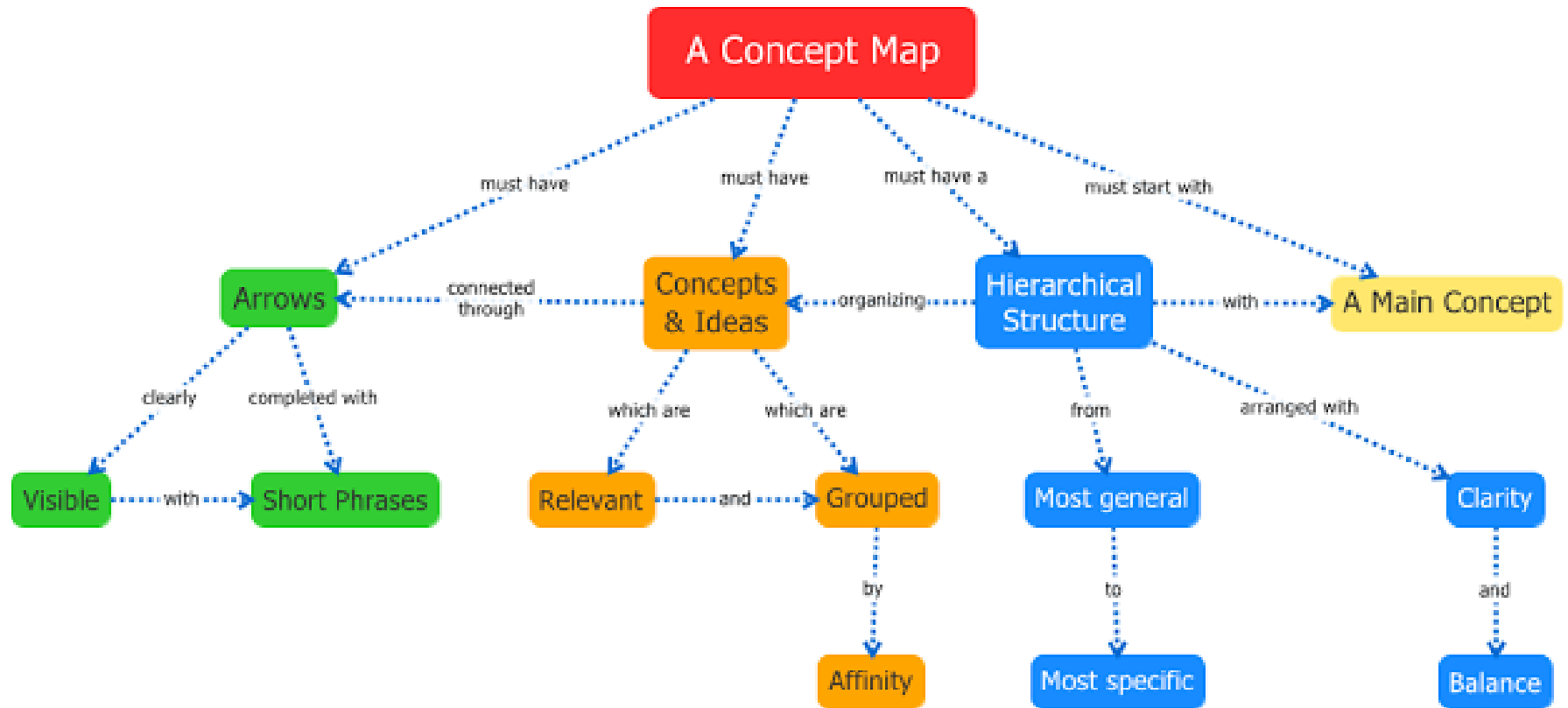
- Pueda **definir conocimiento independientemente de como es usado**.
- La **correctitud del programa está dada por la semántica, no por la implementación particular del control**. Conveniente para performance y optimizaciones.



Conocimiento ontológico

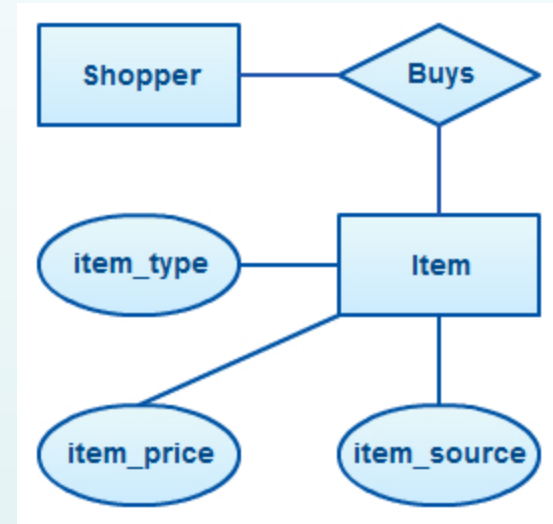
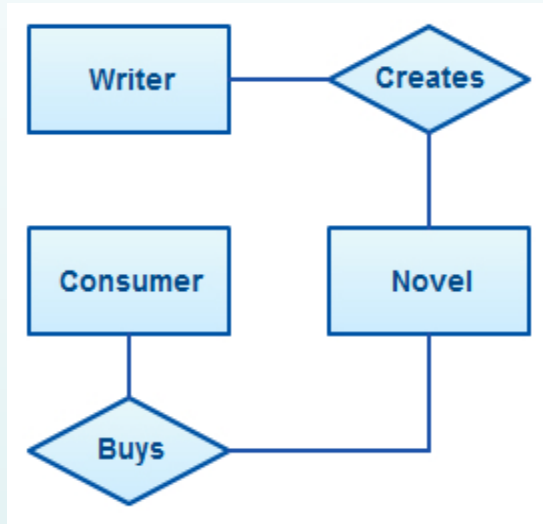
- Una **ontología** es un compromiso a qué es lo que existe en un dominio particular (vista del pequeño mundo que nos interesa).
- Asumimos que el mundo puede representarse en términos de cosas y relaciones entre cosas.
- Cada **base de conocimiento** está asociada a una conceptualización (explícita o implícitamente).
 - Para estos sistemas, lo que **existe** es lo que se **representa**, esto afecta directamente la resolución del problema.
 - Este conjunto de cosas y las relaciones se reflejan en el **vocabulario** de representación con el cual el programa basado en conocimiento representa el conocimiento (vocabulario de discurso).

Ejemplo Conceptualización



2013 Philippe Boukobza
Based on Rémi Bachelet "How to assess a concept map"

Ejemplo Conceptualización



Fuente Imagen Izquierda: <https://creately.com/blog/diagrams/er-diagrams-tutorial/>

Fuente Imagen Derecha: <https://sites.google.com/site/bsiscapstone/capstone-manuscript/chapter4/ultimate-guide-to-er-diagrams>

Representación de conocimiento

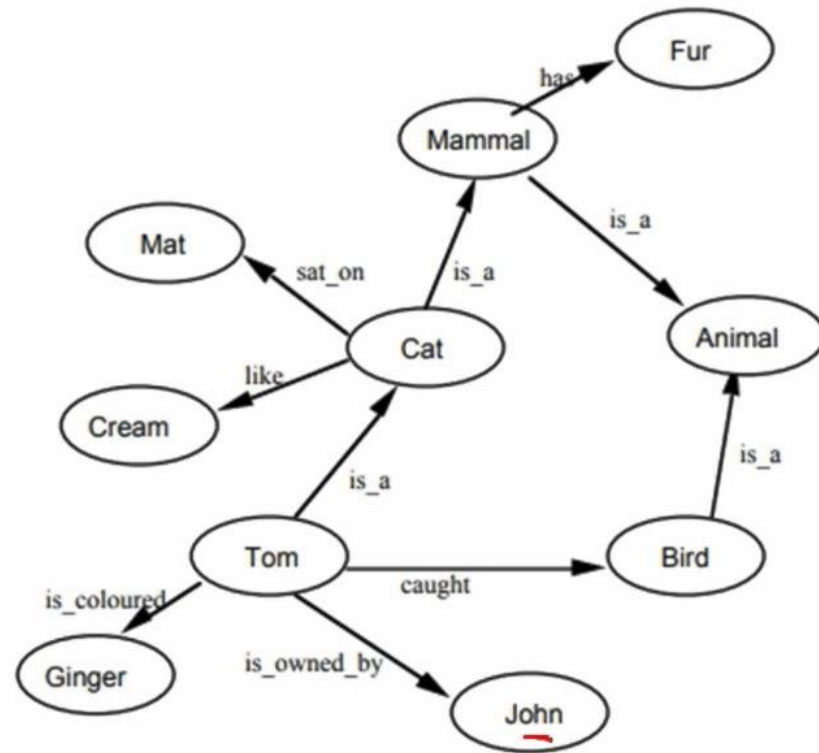
- En el enfoque basado en *lógica*, el lenguaje de representación es en general una variante del cálculo de predicados de primer orden y la tarea de razonar implica la verificación de *consecuencias* lógicas.
- Los enfoques *no lógicos*, se basan en interfaces gráficas y el conocimiento es representado en estructuras de datos *ad hoc*.
 - Redes semánticas [Quillian1967]
 - *Frames* [Minsky1981]
 - Caracterizan el conocimiento y el razonamiento por medio de estructuras cognitivas en forma de redes.

Redes Semánticas

Semantic Networks:

Example :

- ✓ Tom is a cat.
 - ✓ Tom caught a bird.
 - ✓ Tom is owned by John.
 - ✓ Tom is ginger in colour.
- Cats like cream.
The cat sat on the mat.
A cat is a mammal.
A bird is an animal.
All mammals are animals.
Mammals have fur.



Una red semántica es un modelo gráfico para representar conocimiento en patrones de conexión de nodos y arcos interconectados.

Representación de conocimiento

- Los sistemas basados en redes son mas atractivos desde el punto de vista *práctico*.
- Sin embargo, su falta de caracterización *semántica* es un problema importante.
- Ejemplo: UML, no tiene semántica, cada diagrama UML se interpreta de manera distinta una vez que se implementa, cada sistema se comporta de manera diferente aun cuando representen los mismos objetos del mundo real.
- Sin embargo, puede dársele una semántica en *FOL* (al menos a un conjunto central de sus características).
- Las lógicas de descripción surgen de la *combinación* de ambos enfoques.

Lógicas de descripción

- Comenzaron a desarrollarse bajo el nombre de “*Sistemas Terminológicos*”, estableciendo la terminología básica adoptada en el modelamiento de un dominio.
- Luego, el énfasis fue en el conjunto de constructores formadores de *conceptos* admitidos en el lenguaje.
- Más recientemente la atención del I+D en el área se orientó hacia las propiedades de los sistemas lógicos subyacentes y se acuñó el término “Lógicas de Descripción” (*Description Logics* o DLs).

Lógicas de descripción

- Una *familia* de formalismos de representación de conocimiento basados en lógica:
 - Describen el *dominio* de interés en términos de *conceptos* (clases), *roles* (relaciones) e *individuos*.
 - Semántica formal (basada en *teoría de modelos*):
 - Corresponden a fragmentos decidibles de FOL.
 - Relacionadas con Lógicas Proposicionales Modales y Lógicas Dinámicas.

Lógicas de descripción

- Una *familia* de formalismos de representación de conocimiento basados en lógica:
 - Proveen servicios de *inferencia*:
 - Existen procedimientos sanos y completos para problemas específicos de *razonamiento*: inferencia lógica (sentencias atómicas o conjuntivas), respuesta a consultas, inferencia tolerante a la inconsistencia, etc.
 - Sistemas *implementados* con un alto grado de *optimización* que permite resolver problemas de la Web Semántica.

Lógicas de descripción

- Se asumen dos *alfabetos* de símbolos disjuntos:
 - Uno denota los conceptos atómicos – predicados *unarios*.
 - El otro para expresar relaciones entre conceptos (predicados *binarios*).
- El dominio de interpretación es arbitrario y puede ser infinito:
 - La posibilidad de dominios infinitos y la *suposición de mundo abierto* distinguen a las DLs de los lenguajes de bases de datos clásicos (y Datalog también).
- Las características de cada DL están definidas por los *constructores* que establecen relaciones entre objetos.

Lógicas de descripción

- Una Lógica de Descripción se caracteriza por:
 - Un **lenguaje de descripción**: cómo formar conceptos y roles.
$$Human \sqcap Male \sqcap \exists hasChild \sqcap \forall hasChild.(Doctor \sqcup Lawyer)$$
 - Un mecanismo para **especificar conocimiento implícito** acerca de los conceptos y roles (una *TBox* o *axiomas terminológicos*).
$$T = \{ Father \equiv Human \sqcap Male \sqcap \exists hasChild, \\ HappyFather \sqsubseteq Father \sqcap \forall hasChild.(Doctor \sqcup Lawyer) \}$$

Lógicas de descripción

- Una Lógica de Descripción se caracteriza por:
 - Un mecanismo para **especificar propiedades** acerca de los objetos (*ABox* o *axiomas de aserciones*).

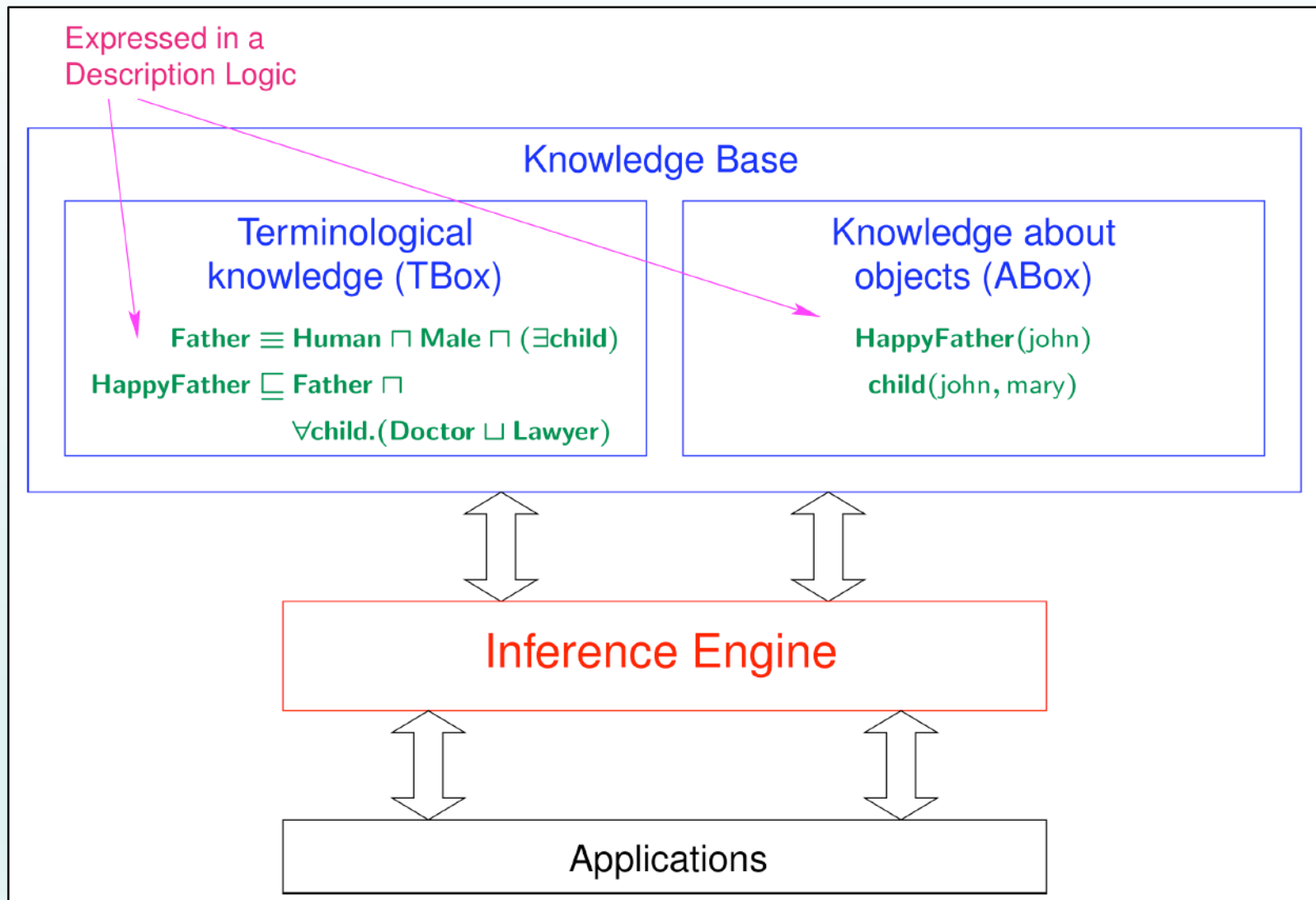
$$A = \{ HappyFather(john), hasChild(john, mary) \}$$

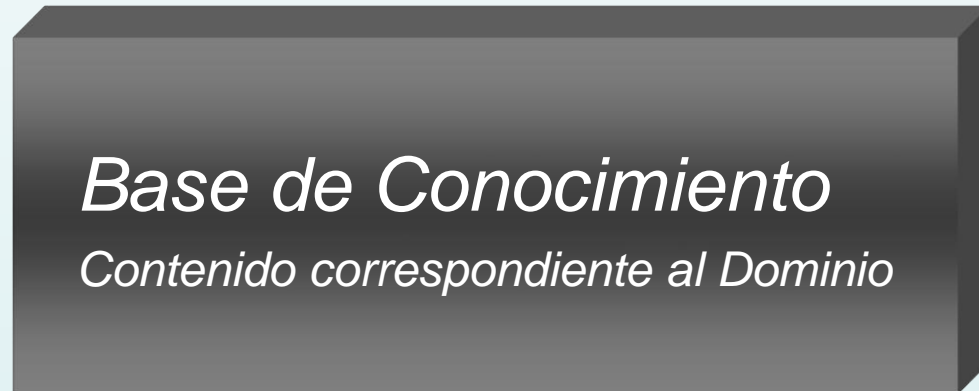
- Un conjunto de **métodos de inferencia**: cómo razonar acerca del conocimiento contenido en una *KB*.

$$T \models HappyFather \sqsubseteq \exists hasChild.(Doctor \sqcup Lawyer)$$

$$T \cup A \models (Doctor \sqcup Lawyer)(mary)$$

Arquitectura de un SRR con DLs





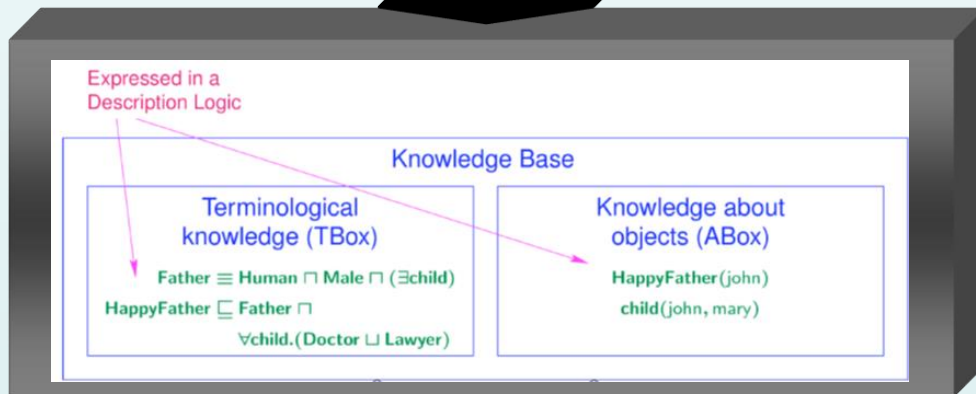


APLICACIONES



Máquina de Inferencia

Algoritmos Independientes del Dominio



DLs: Lenguaje de descripción

- Un lenguaje de descripción se caracteriza por un conjunto de *constructores* para crear conceptos complejos y roles a partir de los básicos:
 - Los conceptos corresponden a *clases*: interpretados como un conjunto de objetos.
 - Roles corresponden a *relaciones*: interpretados como relaciones binarias entre objetos.
- La semántica formal está dada en términos de *interpretaciones* (FOL).

DLs: Semántica (FOL)

- Formalmente: una **interpretación** $I = (\Delta^I, \cdot^I)$ consiste de
 - Un conjunto no vacío Δ^I , con dominio I
 - una función de interpretación \cdot^I , que mapea
 - cada individuo a a un elemento a^I de Δ^I
 - cada concepto atómico A a un subconjunto A^I de Δ^I
 - cada role atómico P a un subconjunto P^I de $\Delta^I \times \Delta^I$
- La función de interpretación se extiende a conceptos complejos y roles de acuerdo con la estructura sintáctica.

DLs: Constructores

Construct	Syntax	Example	Semantics
atomic concept	A	Doctor	$A^I \subseteq \Delta^I$
atomic role	P	hasChild	$P^I \subseteq \Delta^I \times \Delta^I$
atomic negation	$\neg A$	\neg Doctor	$\Delta^I \setminus A^I$
conjunction	$C \sqcap D$	Hum \sqcap Male	$C^I \cap D^I$
(unqual.) exist. res.	$\exists R$	\exists hasChild	$\{ a \mid \exists b. (a, b) \in R^I \}$
value restriction	$\forall R.C$	\forall hasChild.Male	$\{ a \mid \forall b. (a, b) \in R^I \rightarrow b \in C^I \}$
bottom	\perp		\emptyset

C , D denotan conceptos arbitrarios y R un rol arbitrario.

- Estos constructores forman el lenguaje básico AL de la familia de lenguajes AL.

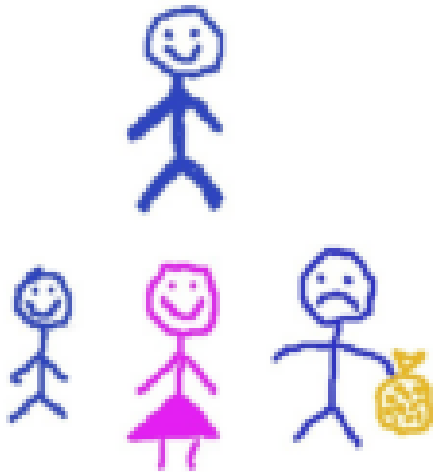
DLs: Constructores

Construct	\mathcal{AL}	Syntax	Semantics
disjunction	\mathcal{U}	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
top		\top	$\Delta^{\mathcal{I}}$
qual. exist. res.	\mathcal{E}	$\exists R.C$	$\{ a \mid \exists b. (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}} \}$
(full) negation	\mathcal{C}	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
number restrictions	\mathcal{N}	$(\geq k R)$	$\{ a \mid \#\{b \mid (a, b) \in R^{\mathcal{I}}\} \geq k \}$
		$(\leq k R)$	$\{ a \mid \#\{b \mid (a, b) \in R^{\mathcal{I}}\} \leq k \}$
qual. number restrictions	\mathcal{Q}	$(\geq k R.C)$	$\{ a \mid \#\{b \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq k \}$
		$(\leq k R.C)$	$\{ a \mid \#\{b \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq k \}$
inverse role	\mathcal{I}	R^{-}	$\{ (a, b) \mid (b, a) \in R^{\mathcal{I}} \}$
role closure	reg	\mathcal{R}^*	$(R^{\mathcal{I}})^*$

- Se han investigado muchos constructores y sus combinaciones, dando lugar a *distintas familias* de DLs con diferente expresividad y complejidad computacional.

Ejemplo DLs

Un **padre contento** es un **hombre** que **tiene al menos** un **hijo hombre** y **al menos** una **hija mujer** y, que **todos** sus hijos son **contentos** o **ricos**.



Conceptos = { Hombre, Mujer, Contento, Rico }

Roles = { tiene-hijo }

Individuos = { carlos }

PadreContento \equiv Hombre \sqcap

\exists tiene-hijo.Hombre \sqcap

\exists tiene-hijo.Mujer \sqcap

\forall tiene-hijo.(Contento \sqcup Rico)

carlos: \neg PadreContento

Otro ejemplo DLs

La T-Box:

Mujer	\sqsubseteq	Persona $\sqcap \exists \text{sexo.Femenino}$
Hombre	\sqsubseteq	Persona $\sqcap \exists \text{sexo.Masculino}$
PadreOMadre	\equiv	Persona $\sqcap \exists \text{hijo-de.Persona}$
Madre	\equiv	Mujer \sqcap PadreOMadre
Padre	\equiv	Hombre \sqcap PadreOMadre

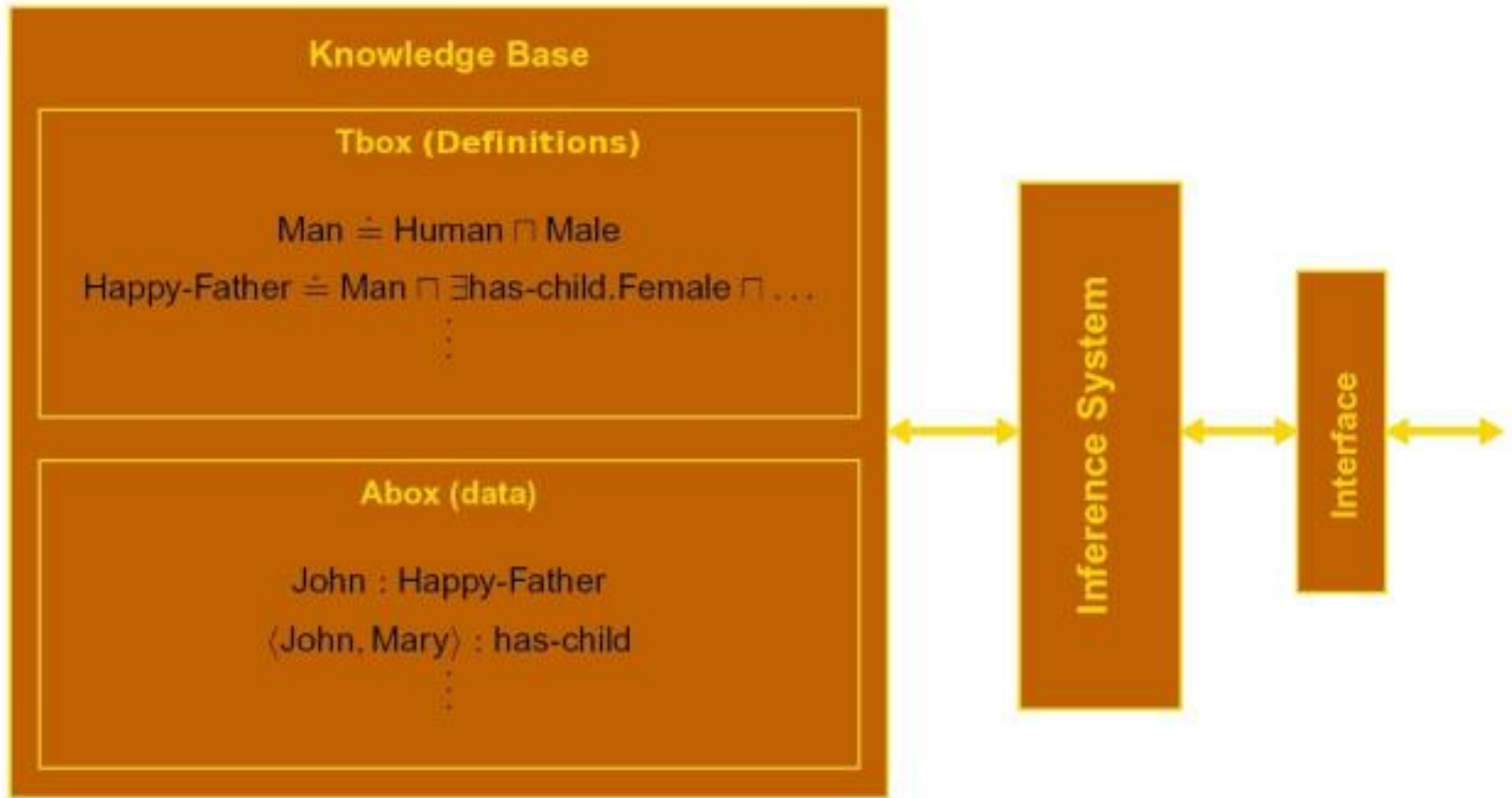
La A-Box:

alicia:Madre

(alicia,betty):hijo-de

(alicia,carlos):hijo-de

Arquitectura para DLs



Ejercicio 1: DLs

Definir los siguientes conceptos:

- “abuela materna”
- “tío sin hijos”
- “sobrino de una nuera”
- “padre con al menos 3 hijos, dos de los cuales son mujeres”

Puede usar conceptos intermedios pero debe definirlos en términos de los conceptos definidos en la slide 28.

DLs: Razonamiento / Consultas

- El problema de razonamiento clásico en DLs es *inclusión de conceptos* (*concept subsumption*), denotado $C \sqsubseteq D$:
 - Se chequea si un concepto D se considera *más general* que el concepto $C \Rightarrow$ si C siempre denota un subconjunto de lo que denota D .
- Otro tipo de razonamiento típico en DLs es *satisfacción de conceptos*, es el problema de chequear si un concepto puede denotar un conjunto *no vacío* (concepto vacío).
 - **Satisfabilidad** es un caso especial de inclusión de conceptos, asumiendo que D es el conjunto vacío (concepto insatisfacible) $\Rightarrow C \sqsubseteq \emptyset$?

El rol de las ontologías

- Las ontologías probablemente jueguen un rol importante en el proceso de hacer que la información on-line sea ‘**comprensible**’ para **acceso automático**.
- Facilita el **reúso** e **intercambio** de conocimiento: fuente de definiciones **precisas** de **términos** que pueden ser **compartidas** entre aplicaciones (**SNOMED**).
- Cuanto mayor sea el grado de **formalización** y mayor **regularidad**, más **fácil** va a ser que la ontología sea **manipulable** automáticamente.

Áreas de aplicación DLs

Bases de conocimiento terminológicas y ontologías

- Especialmente útiles como lenguaje de definición y mantenimiento de ontologías

Aplicaciones en Bases de Datos

- DLs pueden capturar la semántica de varias metodologías de modelado en BD e.g., diagramas de ER, UML, etc., y así, proveer soporte durante el diseño de diagramas, mantenimiento y consulta.
- Integración de BD: integración e intercambio de datos.
- Bases de datos federadas.
- Ontology Based Data Access (OBDA) - OMQA

Áreas de aplicación

Web Semántica (la tercer evolución de la Web):

- Los recursos on-line (ya no simples páginas HTML) deben ser accesibles más fácilmente por **procesos automáticos**.
- La propuesta es alcanzar este objetivo mediante:
 - Agregar '**markup semántico**' a la información en la web, que definirían ontologías con una semántica clara.
 - **Metadata** (anotaciones adicionales al contenido actual de la página) que especifican contenido/función.
 - **Knowledge graphs** = Ontologías + ML
- **Desarrollo, mantenimiento y fusión** de estas ontologías y para la **evaluación dinámica** de recursos (e.g., búsqueda).

Lenguajes basados en Reglas



Lenguajes basados en Reglas

- Diseñado para bases de datos *deductivas*:
 - Bases de datos que permiten obtener información que está contenida *implícitamente* de manera deductiva (introduce recursión).
 - 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*): no hay funciones ni negación.



Datalog: Intuitivamente

- Ejemplo : clausura C de una relación binaria r

- $C(X,Y) \leftarrow r(X, Y)$
- $C(X,Z) \leftarrow C(X,Y) \wedge r(Y,Z)$

Si en mi instancia de base de datos tengo información de la forma $r(X,Y)$ entonces puedo derivar nuevas conclusiones de la forma $C(X,Y)$.

- Reglas de la forma $H \leftarrow B_1, B_2, \dots B_n$
- En FOL es equivalente a: $\forall \mathbf{X} \forall \mathbf{Y} \phi(\mathbf{X}, \mathbf{Y}) \rightarrow R(\mathbf{X})$

Datalog: Syntax

- ❖ Un universo infinito de **constantes** Δ
- ❖ 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;
 - Usamos \mathbf{X} para denotar la secuencia X_1, \dots, X_n , $n \geq 0$.
 - Si $p \in \mathcal{R}$, de aridad n , un **átomo** es de la forma $p(X_1, \dots, X_n)$, si X_1, \dots, X_n son constantes **es átomo básico**.

Datalog: Syntax reglas

- Una regla en Datalog tiene la siguiente forma:

$$\Psi(\mathbf{X}) \leftarrow \Phi(\mathbf{X}, \mathbf{Y})$$

donde $\Phi(\mathbf{X}, \mathbf{Y})$ es una conjunción de átomos y $\Psi(\mathbf{X})$ un átomo.

- Llamaremos a $\Psi(\mathbf{X})$ la cabeza o conclusión y a $\Phi(\mathbf{X}, \mathbf{Y})$ el cuerpo de la regla.
- Si el cuerpo es vacío, entonces “ $\Psi(\mathbf{X}) \leftarrow$ ” denota un hecho (omitiremos la flecha).
- Generalizan las dependencias de inclusión de las bases de datos: ej., foreign keys.
- Un programa Datalog es un conjunto de reglas.

Datalog: Ejemplo

father(alice, bob).

mother(alice, carla).

mother(evan, carla).

father(carla, david).

parent(X, Y) ← father(X, Y)

parent(X, Y) ← mother(X, Y)

ancestor(X, Y) ← parent(X, Y)

ancestor(X,Z) ← parent(X, Y) ∧ ancestor(Y,Z)

Datalog: Semántica Deductiva

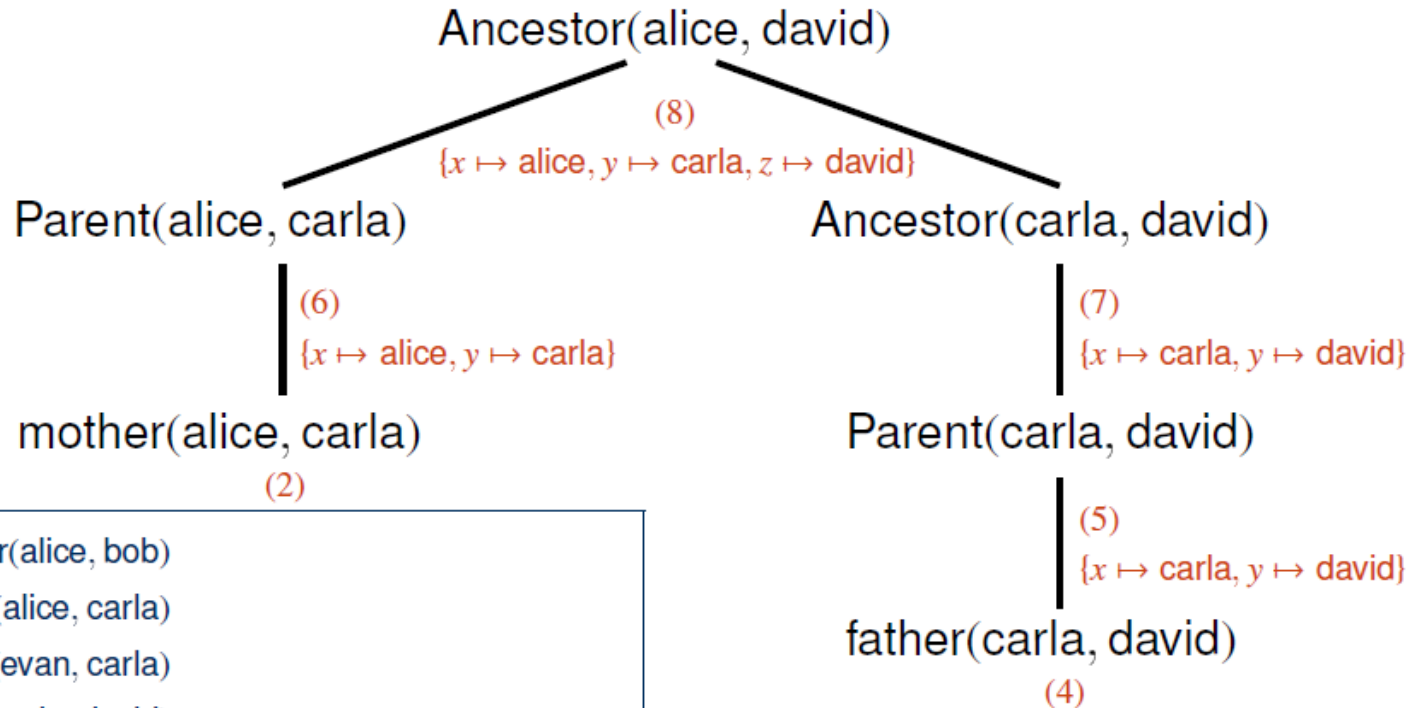
- En general nos interesan consultas sobre que átomos básicos se derivan de un programa Datalog.
- Restringimos al conjunto de constantes que aparecen en el programa (finito)
- Las variables pueden representar constantes arbitrarias de este conjunto: una **sustitución básica** mapea variables a constantes.
- Una regla **es aplicable** a un conjunto A de átomos básicos, el cuerpo de una regla es satisfecho en A si y solo si existe una sustitución básica θ tal que θ aplicada al cuerpo está en A .

Ejemplo: La regla $parent(X, Y) \leftarrow mother(X, Y)$ puede aplicarse al átomo $mother(alice, carla)$ bajo la sustitución $\{X/alice, Y/carla\}$.

Datalog: Semántica Deductiva - Formal

- Si una regla es aplicable bajo alguna sustitución básica, entonces la **instancia que resulta de aplicar la sustitución a la cabeza de la regla** es **derivada** (entailed/modelada) por el programa.
- Sea P un programa Datalog, el conjunto de átomos que pueden derivarse de P es el **menor conjunto** de átomos A para el cual existe una regla $r: H \leftarrow B_1 \wedge \dots \wedge B_n$ y una sustitución básica θ tal que:
 - $A = H\theta$, y
 - Para cada $i \in \{1, \dots, n\}$, $B_i\theta$ se puede derivar de P .

Datalog: Árbol de derivación



- (1) father(alice, bob)
- (2) mother(alice, carla)
- (3) mother(ewan, carla)
- (4) father(carla, david)
- (5) Parent(x, y) ← father(x, y)
- (6) Parent(x, y) ← mother(x, y)
- (7) Ancestor(x, y) ← Parent(x, y)
- (8) Ancestor(x, z) ← Parent(x, y) ∧ Ancestor(y, z)

Fuente: Database Theory – Introduction to Datalog <https://iccl.inf.tu-dresden.de/w/images/b/b8/DBT2019-Lecture-12-overlay.pdf>

Ejercicio 2

Dado el programa Datalog anterior, indique si:

- Se puede derivar $parent(evan, david)$?
- Se puede derivar $ancestor(carla, david)$?

En el caso donde se pueda muestre el árbol de derivación.

Datalog: Menor punto fijo

- El **programa básico** (ground) de un programa Datalog P (denotado $ground(P)$) es el conjunto de reglas básicas que se obtienen de las reglas de P remplazando uniformemente sus variables con constantes del universo.
- Las derivaciones se pueden describir con un **operador de consecuencias inmediato** TP que mapea un conjunto de hechos básicos I a conjuntos de hechos básicos $TP(I)$:
$$TP(I) = \{H \mid H \leftarrow B_1 \wedge \dots \wedge B_n \in ground(P) \text{ y } B_1 \wedge \dots \wedge B_n \in I\}$$
- Menor punto fijo (LFP) de TP : es el conjunto de átomos básicos L más chico tal que $TP(L) = L$

Datalog: Menor punto fijo

- Computación de encadenamiento hacia adelante (forward chaining):

$$T^0_P = \emptyset \text{ y } T^{i+1}_P = TP(T^i_P)$$

El menor punto fijo de TP es $T^\infty_P = \bigcup_{i \geq 0} T^i_P$

Derivación: Un átomo A básico se deriva de un programa

Datalog P si y solo si $A \in T^\infty_P$

- Ver encadenamiento hacía atrás (SLD Resolution) en Cap. 2.7 “Computational Intelligence: A Logical Approach - D.Poole, A. Mackworth, R. Goebel Oxford University Press.”

Datalog: Menor punto fijo

Si leemos una regla Datalog en FOL:

$$parent(X, Y) \leftarrow mother(X, Y)$$

$$\forall X, Y (mother(X, Y) \rightarrow parent(X, Y))$$

- Dado un programa Datalog P sea F_P el conjunto de implicaciones de FOL que P representa.
- Un conjunto de implicaciones de FOL puede tener muchos modelos:
 - Consideremos el mínimo modelo de F_P sobre el dominio definido por el universo del programa Datalog P
 - **El menor punto fijo de P coincide con el mínimo modelo de F_P**

Datalog como lenguaje de consulta

- Podemos usar Datalog como lenguaje de consulta de una instancia de bases de datos relacional.
 - La instancia de bases de datos es un conjunto de átomos básicos
 - Debemos especificar el predicado de la respuesta.
- Una consulta Datalog es un par $\langle R, P \rangle$, donde P es un programa Datalog y R es un predicado de respuesta. El resultado de la consulta es el conjunto de hechos de la forma de R que se derivan de P .

Consulta Datalog: Ejemplo

Sea D una instancia de base de datos:

$$D = \{father(alice, bob), mother(alice, carla), \\ mother(ewan, carla), father(carla, david)\}$$

Sea P el siguiente programa Datalog:

$$P = \{parent(X, Y) \leftarrow father(X, Y), parent(X, Y) \leftarrow mother(X, Y), \\ ancestor(X, Y) \leftarrow parent(X, Y), \\ ancestor(X, Z) \leftarrow parent(X, Y) \wedge ancestor(Y, Z)\}$$

Consulta Datalog: Ejemplo

- Dada la consulta $Q = \langle R, P \rangle$ con R es *parent*:

El conjunto de respuesta a Q es:

$$\{parent(alice, bob), parent(alice, carla), \\ parent(evan, carla), parent(carla, david)\}$$

Importante:

- Identificamos dos tipos de predicados IDB y EDB (intencionales y extensionales resp.)
- Los predicados que aparecen en cabezas de reglas en P son **intencionales**.
- Los predicados que solo parecen en el cuerpo de reglas de P son **extensionales**.
- Las relaciones en la BD se usan solo como predicados EDB.



Consulta Conjuntivas

- Una **consulta conjuntiva** (CQ) sobre un conjunto de predicados \mathcal{R} tiene la forma $Q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$, Φ es una conjunción de átomos. Las variables en \mathbf{X} son las variables libres en Φ que se esperan instanciar en las respuestas.
- 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. La respuesta es **verdadero** o **falso**.

Consulta Conjuntivas en Datalog

Dada una consulta conjuntiva:

$$Q(X_1, \dots, X_n) = \exists Y_1, \dots, Y_m. A_1 \wedge \dots \wedge A_k$$

se puede expresar como una consulta Datalog $\langle Q, P \rangle$ donde P tiene la regla

$$Q(X_1, \dots, X_n) \leftarrow A_1 \wedge \dots \wedge A_k$$

Ejemplo: dada la consulta conjuntiva:

$$Q(X) = \exists Y. mother(X, Y)$$

(Q pide el conjunto de personas que tienen madre, en la respuesta no nos importa la madre)

Le agregamos a P la regla:

$$Q(X) \leftarrow mother(X, Y)$$

El conjunto de respuesta será: $\{Q(alice), Q(evan)\}$

ABCDatalog

- AbcDatalog es una implementación open-source para Datalog en Java.
- Créditos: Harvard University bajo el financiamiento de National Science Foundation Nos. 1237235 and 1054172.
- Se puede usar su API desde JAVA y tiene una interfaz gráfica disponible.
- Instalación y manual en: <https://abcdatalog.seas.harvard.edu/>

Más Ejercicios

Ejercicio 3: Dada la consulta conjuntiva:

$$Q(X,Y) = \exists Z. \textit{parent}(X,Y) \wedge \textit{parent}(Z,Y)$$

¿Qué regla debemos agregarle a P (slide 49)?

¿Cual sería el conjunto de respuesta? Verifíquelo en ABCDatalog.

Ejercicio 4: ¿Cómo sería la regla que hay que agregar dada la consulta

Booleana: $Q1() = \exists Z. \textit{parent}(X,Y) \wedge \textit{parent}(Z,Y)$

Ejercicio 5: Escriba un programa Datalog para resolver el problema del mundo de bloques planteado en el práctico 3. Generalice su programa de manera que se pueda hacer las siguientes consultas: para cualquier bloque X, Y, “está el bloque X sobre la mesa”?, “está X apilado sobre Y”?, “está X debajo de Y”?, “esta X inmediatamente sobre Y”?, “esta X inmediatamente por debajo de Y”?

Pruebe la correctitud del programa utilizando la herramienta ABCDatalog.



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:

$$cientifico \sqsubseteq \exists esAutor de$$

- Conceptos *disjuntos*:

$$artRevista \sqsubseteq \neg artConferencia$$

- *Funciones*: (*funct tienePrimerAutor*) (e.j, dependencia funcional)
- 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 $sen-emp \times emp \sqsubseteq moreThan$	$sen-emp(X), emp(Y) \rightarrow moreThan(X, Y)$
(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 $emp \sqcap customer \sqsubseteq \perp$	$emp(X), customer(X) \rightarrow \perp$
Functionality $func(reports)$	$reports(X, Y), reports(X, Z) \rightarrow Y = Z$

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[∃] (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 por cuestiones de eficiencia.
- 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 **nullos** (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.**
- Igual que antes: una (instancia de) **base de datos** D sobre \mathcal{R} es un conjunto de átomos con predicados en \mathcal{R} y argumentos en Δ .



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.
- Las *respuestas* a una consulta en Datalog+/- se definen vía **homomorfismos**, mapeos $\mu: \Delta \cup \Delta_N \cup \mathcal{V} \rightarrow \Delta \cup \Delta_N \cup \mathcal{V}$:
 - 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** sobre la base de datos $Q(D)$:
conjunto de tuplas t sobre Δ t.q. $\exists \mu: \mathbf{X} \cup \mathbf{Y} \rightarrow \Delta \cup \Delta_N$ t.q.
 $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, y $\mu(\mathbf{X}) = t$.



Datalog+/-: Sintaxis

- **Tuple-generating Dependencies** (TGDs) son restricciones de la forma $\sigma: \forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$ 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 un esquema de base de datos R , una **ontología** **Datalog+/-** es un par (D, Σ) donde D es una instancia de base de datos en R y Σ es un conjunto de TGDs sobre R .

Datalog+/-: Semántica y Consultas

- 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 $B \in mods(D, \Sigma)$.

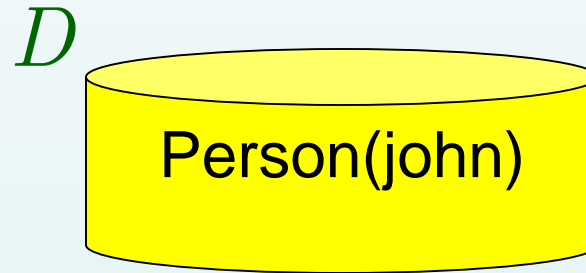
Procedimiento de Inferencia: Chase

- 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 Σ

Output: Un modelo de $D \cup \Sigma$



Σ

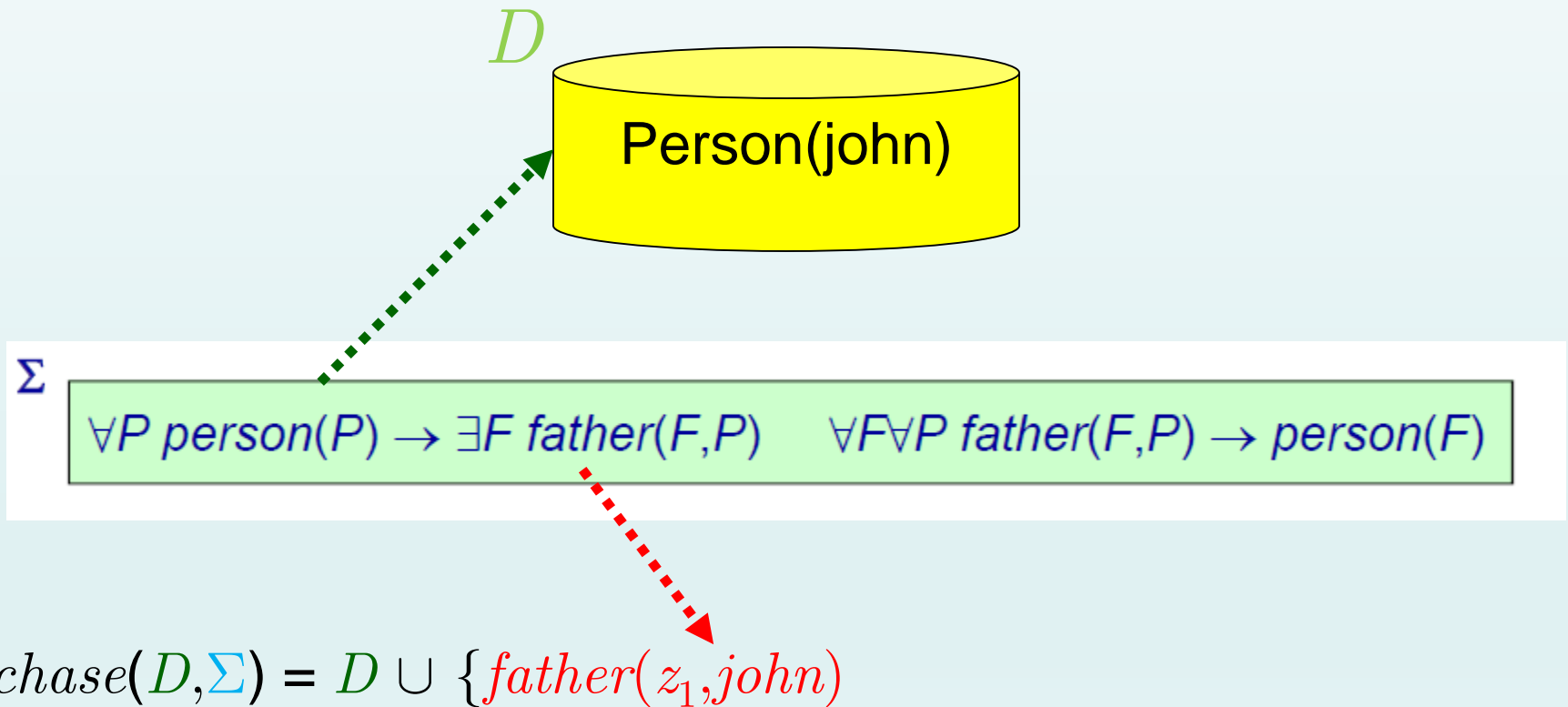
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

$\text{chase}(D, \Sigma) = D \cup ?$

Chase

Input: Base de datos D , conjunto de TGDs Σ

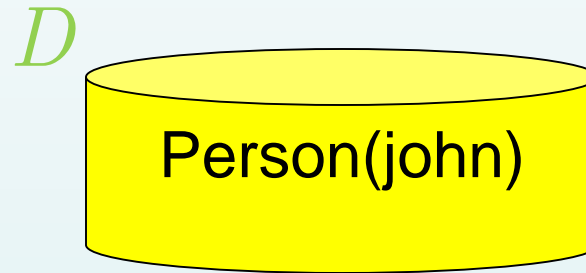
Output: Un modelo de $D \cup \Sigma$



Chase

Input: Base de datos D , conjunto de TGDs Σ

Output: Un modelo de $D \cup \Sigma$



Σ

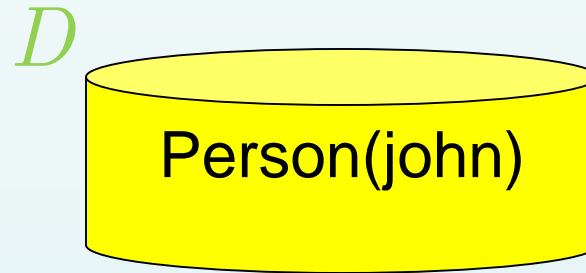
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

$\text{chase}(D, \Sigma) = D \cup \{ \text{father}(z_1, \text{john}), \text{person}(z_1) \}$

Chase

Input: Base de datos D , conjunto de TGDs Σ

Output: Un modelo de $D \cup \Sigma$



Σ

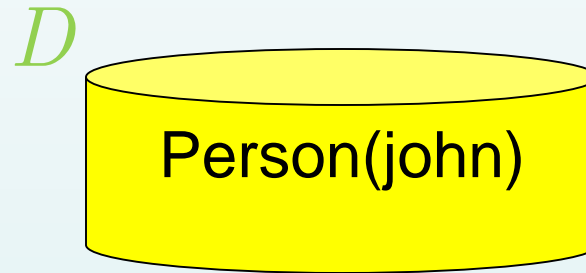
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

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

Chase

Input: Base de datos D , conjunto de TGDs Σ

Output: Un modelo de $D \cup \Sigma$



Σ

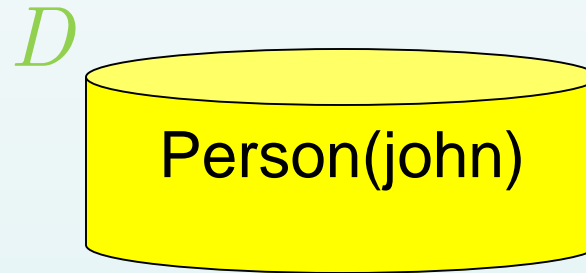
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

$\text{chase}(D, \Sigma) = D \cup \{ \text{father}(z_1, \text{john}), \text{person}(z_1), \text{father}(z_2, z_1), \dots \}$

Chase

Input: Base de datos D , conjunto de TGDs Σ

Output: Un modelo de $D \cup \Sigma$



Σ

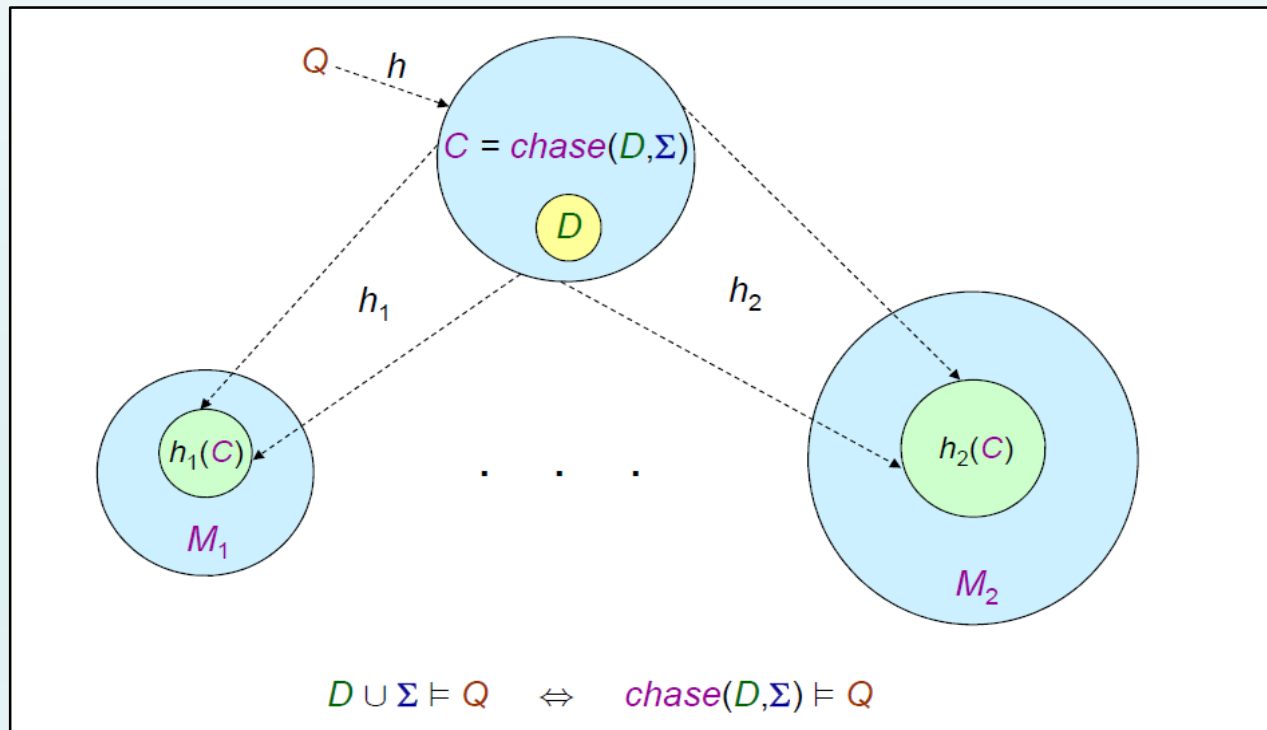
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

$\text{chase}(D, \Sigma) = D \cup \{ \text{father}(z_1, \text{john}), \text{person}(z_1), \text{father}(z_2, z_1), \dots \}$

INSTANCIA INFINITA

Query Answering vía el chase

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



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 Σ .
- *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} .
- En general hacen que responder consultas sobre la ontología sea indecidible: se asume un conjunto de EGDs **separables** (las EGDs y TGDs son independientes entre sí).

Datalog+/-: Ejemplo

$$D = \{ \text{directs}(\text{john}, \text{sales}), \text{directs}(\text{anna}, \text{sales}), \\ \text{directs}(\text{john}, \text{finance}), \text{supervises}(\text{anna}, \text{john}), \\ \text{works_in}(\text{john}, \text{sales}), \text{works_in}(\text{anna}, \text{sales}) \}$$

$$\Sigma_T = \{ \text{works_in}(X, D) \rightarrow \text{emp}(X), \\ \text{manager}(X) \rightarrow \exists Y \text{supervises}(X, Y), \\ \text{supervises}(X, Y) \wedge \text{directs}(X, D) \rightarrow \text{works_in}(Y, D) \}$$


$$\Sigma_{NC \cup EGD} = \{ \text{supervises}(X, Y) \wedge \text{manager}(Y) \rightarrow \perp, \\ \text{supervises}(X, Y) \wedge \text{works_in}(X, D) \wedge \text{directs}(Y, D) \rightarrow \perp, \\ \text{directs}(X, D) \wedge \text{directs}(X, D') \rightarrow D = D' \}$$



Guarded Datalog+/-

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

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

guard 

- El *chase* tiene **treewidth finito** \Rightarrow query answering decidable
- Query answering es **PTIME-completo** en complejidad de datos (TGDS y consulta fija).
- Extiende la Lógica de descripción **ELH** (misma complejidad data).

Guarded Datalog+/-

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

EL TBox	Datalog [±] 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.

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

trivialmente guarded

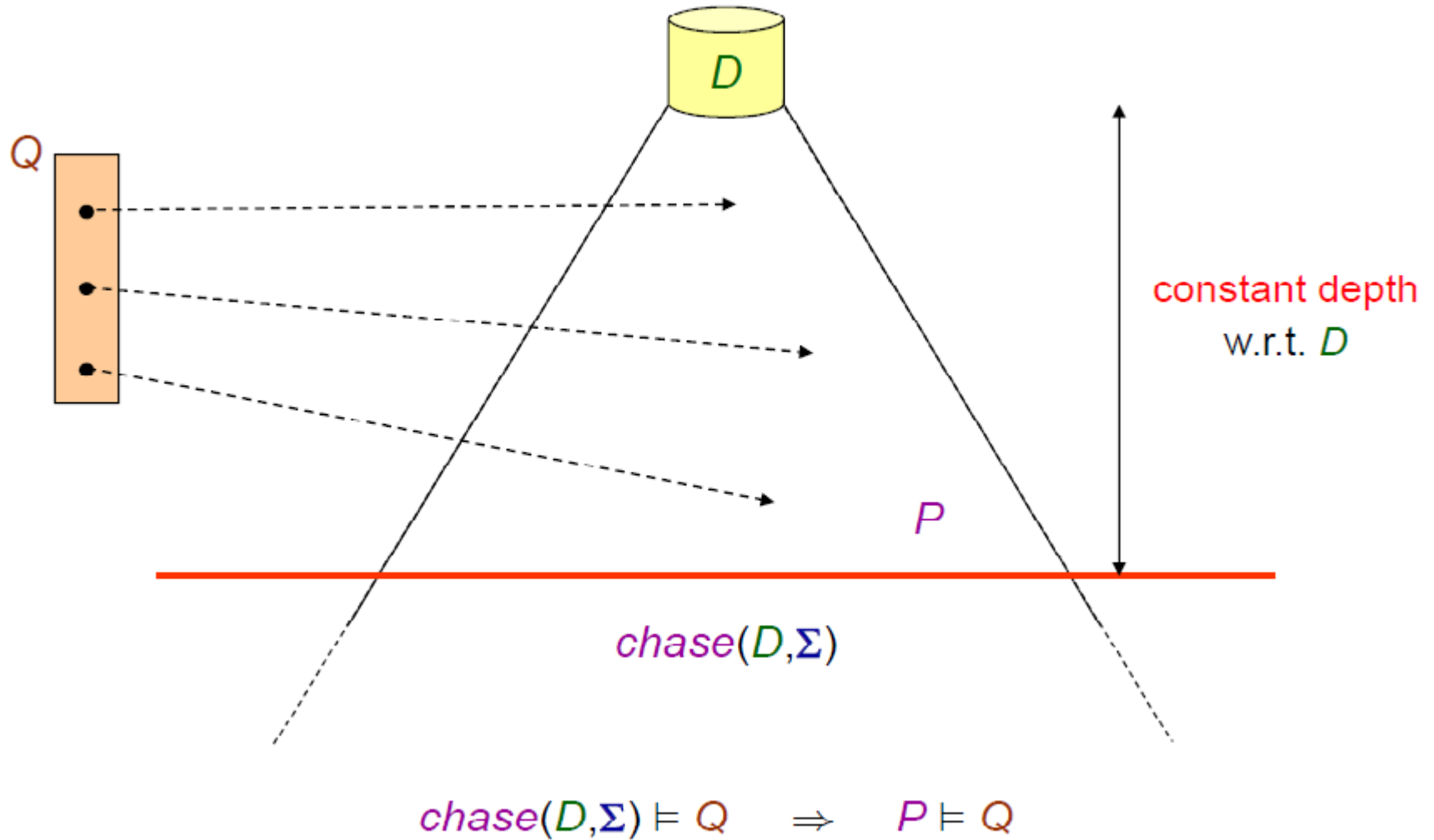
- Las linear TGDs son (trivialmente) *guarded*.
- Query answering está en AC_0 en complejidad de datos (*reescritura de primer orden – FO rewritability – Muy eficientes!!!!*).
- Extiende la (familia de) lógicas de descripción *DL-Lite* (misma complejidad data).

Linear Datalog+/-

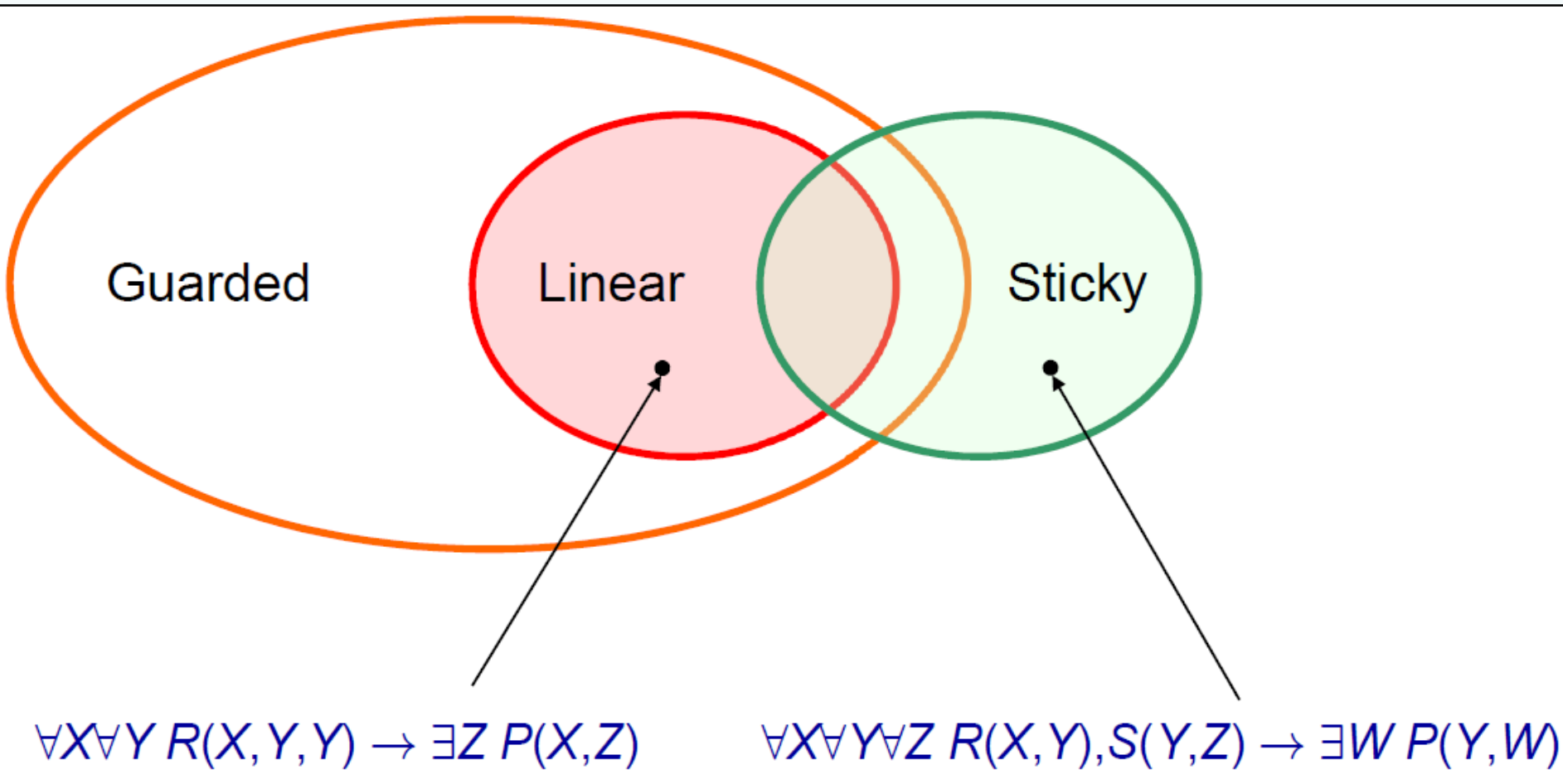
- **DL-Lite** familia de lógicas de descripción con data complejidad AC_0 (OWL 2 QL).

DL-Lite TBox	Datalog [±] Representation
$A \sqsubseteq B$	$\forall X A(X) \rightarrow B(X)$
$A \sqsubseteq \exists R$	$\forall X A(X) \rightarrow \exists Y R(X, Y)$
$\exists R \sqsubseteq A$	$\forall X \forall Y R(X, Y) \rightarrow A(X)$
$R \sqsubseteq P$	$\forall X \forall Y R(X, Y) \rightarrow P(X, Y)$

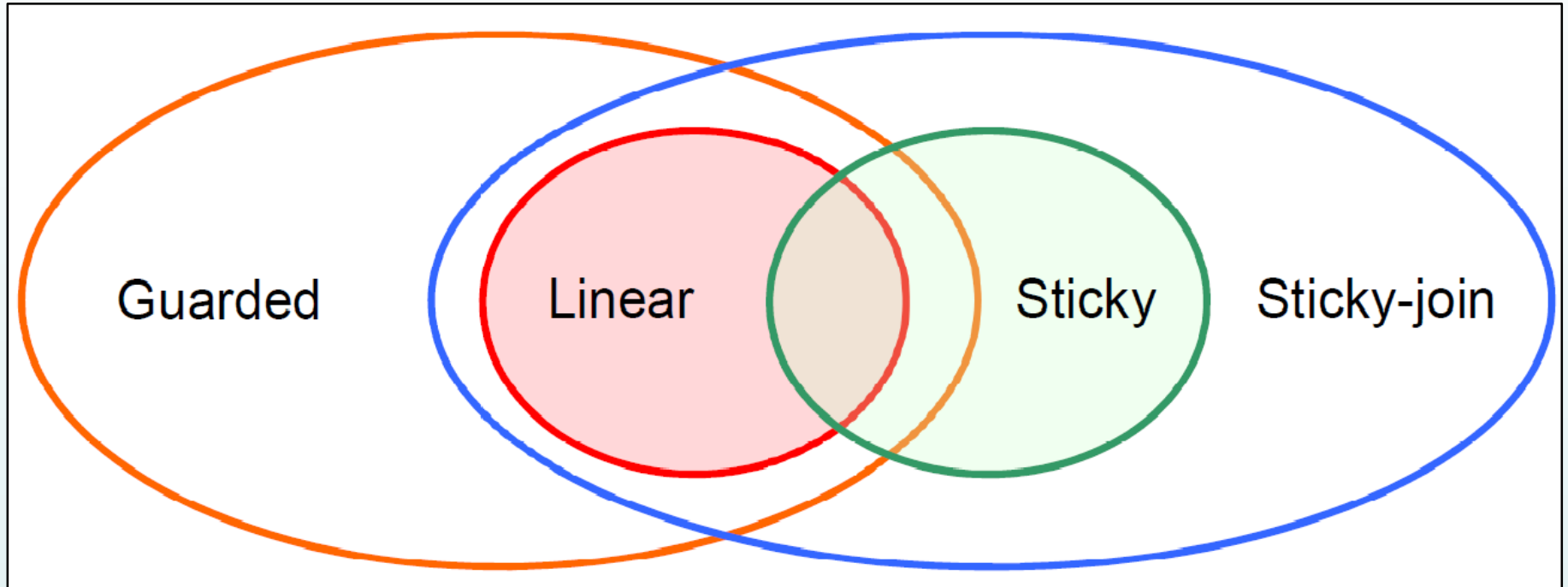
Linear Datalog+/-: chase



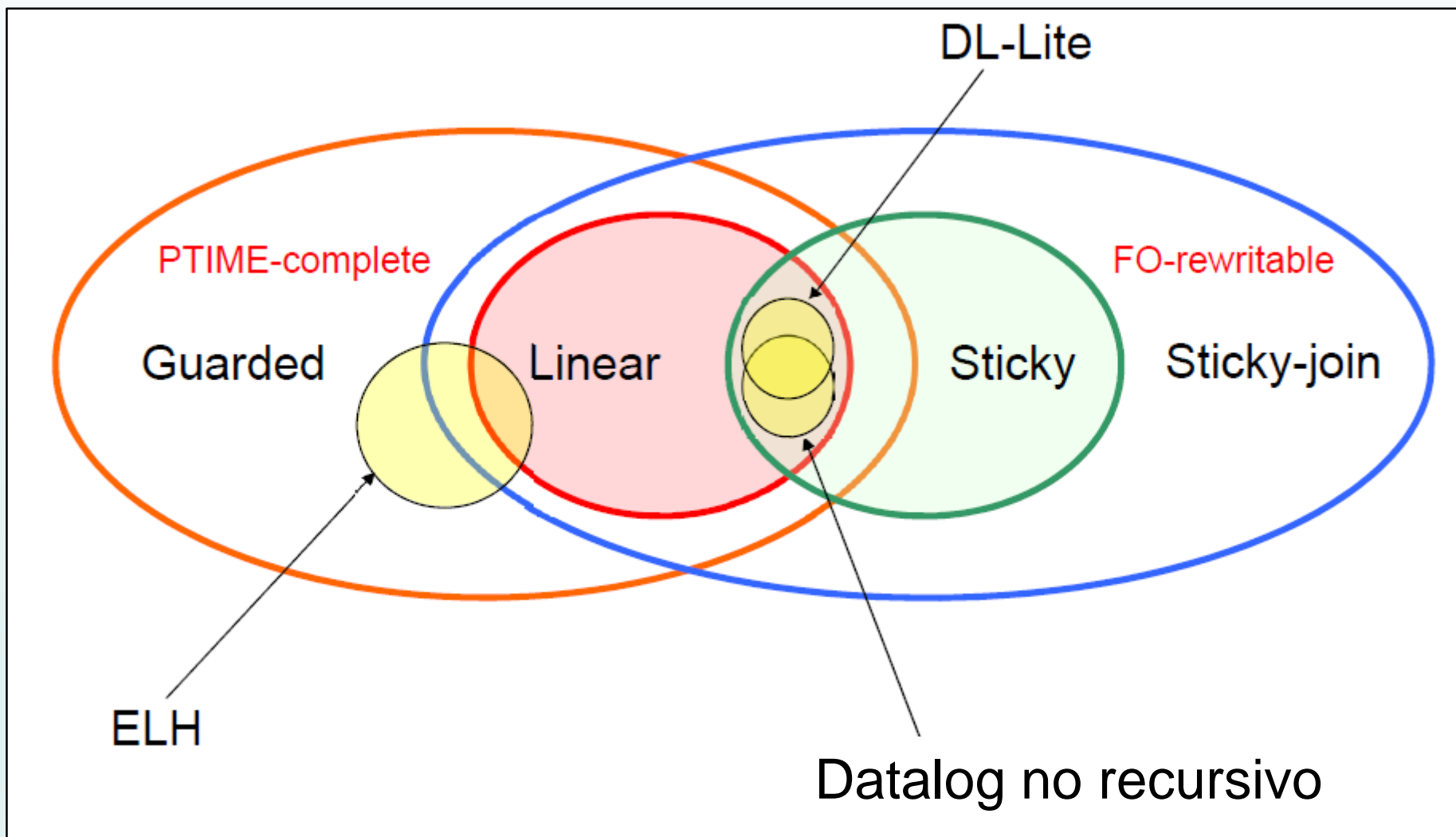
Datalog+/-: Resumen



Datalog+/-: Resumen



Datalog+/-: Resumen



Aplicación de Ontologías: Interoperabilidad entre bases de datos

Ejemplo: Fragmento de una tabla relacional de un sistema de información de bancos

CUC	TS_START	TS_END	ID_GRUP	FLAG_CP	FLAG_CF	FATTURATO	FLAG_FATT	
124589	30-lug-2001					195000,00	N	
140904	15-mag-2001	15-giu-2005	55000	N	N	230600,00	N	
124589	5-mag-2001	30-lug-2004	92736	N	S	195000,00	S	
-452901	13-mag-2001	27-lug-2004	92770	S	N	392000,00	N	
129008	10-mag-2001	1-gen-9999	62010	N	S	247000,00	S	

Valor negativo indica un retiro de dinero

El problema: Integración

- Este ejemplo muestra que en los sistemas del **mundo real**, el significado de los datos en las tablas puede ser **ambiguo**.
- Es crucial entender el **significado** de los datos si queremos manejar de manera “**correcta**” la información en las tablas y extraerla.
 - Fuertemente ligado a como los datos se usan regularmente y poder entenderlo requiere de la **experticia de dominio** (**background knowledge**) los usuarios que lo consumen.
- Además...en general, los sistemas de información usan diferentes fuentes de datos **heterogéneas**, internas y externas a la organización.

Query answering sobre múltiples fuentes

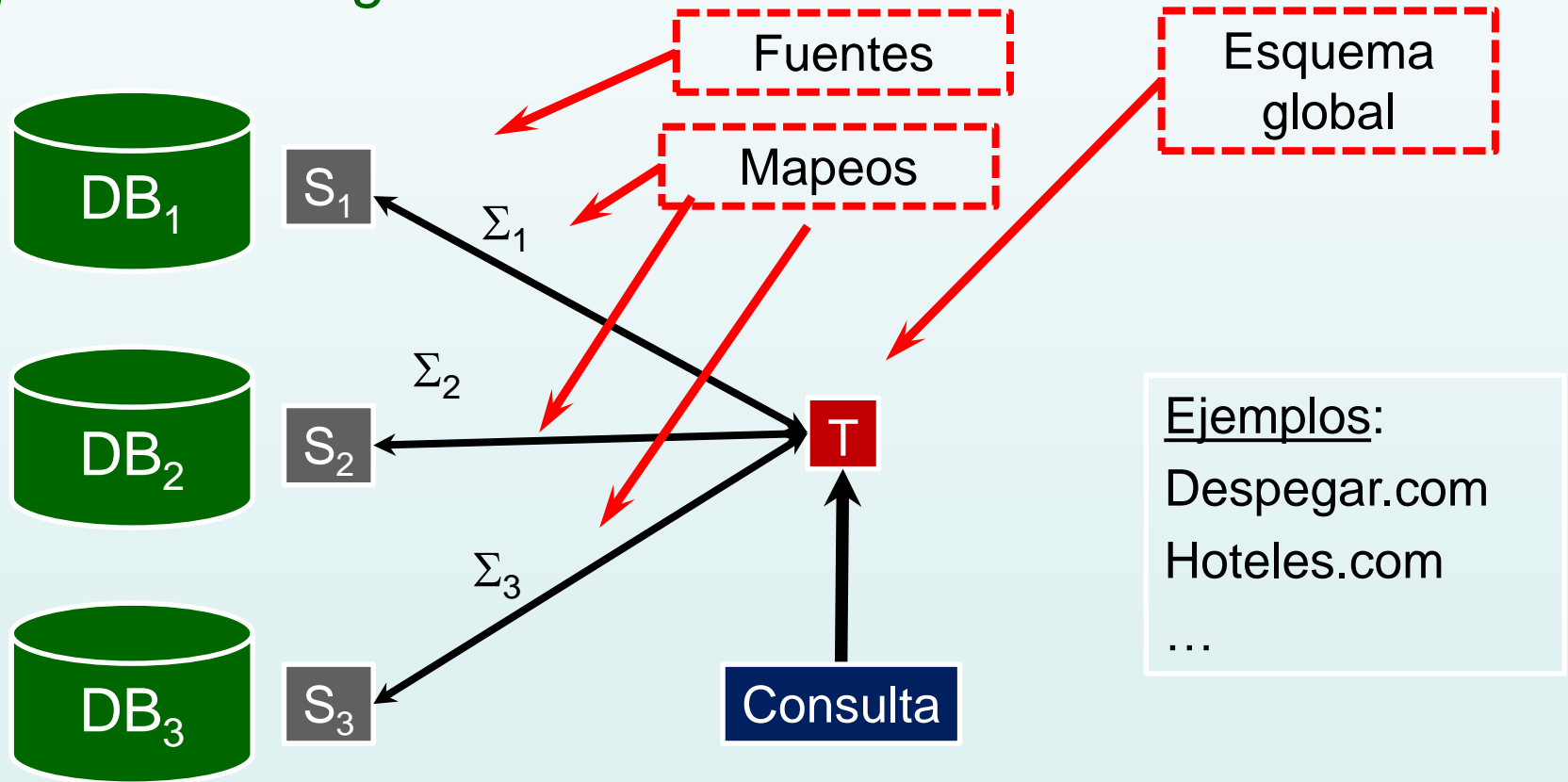
- Datos Distribuidos: residen en diferentes bases de datos
- Datos heterogéneos: pueden tener distintas estructuras/modelos, diferentes políticas de acceso, etc.
- Nuestra vista del mundo (o de un fenómeno o negocio) puede ser muy distinta de la vista subyacente en las bases que necesitamos acceder (diferente semántica).
- Puede que solo parte de los datos estén disponibles.
- Resumen: las fuentes no conforman con el esquema de la BD el la cual vamos a cargar los datos.

Dos propuestas diferentes

- La comunidad de investigación ha estudiado dos facetas diferentes pero estrechamente relacionadas de la integración de información:
 - *Integración* de datos, también llamada Federación de datos (“*Data Integration*” y “*Data Federation*”, en inglés)
 - *Intercambio* de datos, también llamada Materialización o Traducción de datos (“*Data Exchange*” y “*Data Translation*”, en inglés)
- Veamos cómo se caracteriza cada una...

Integración de datos (Federación)

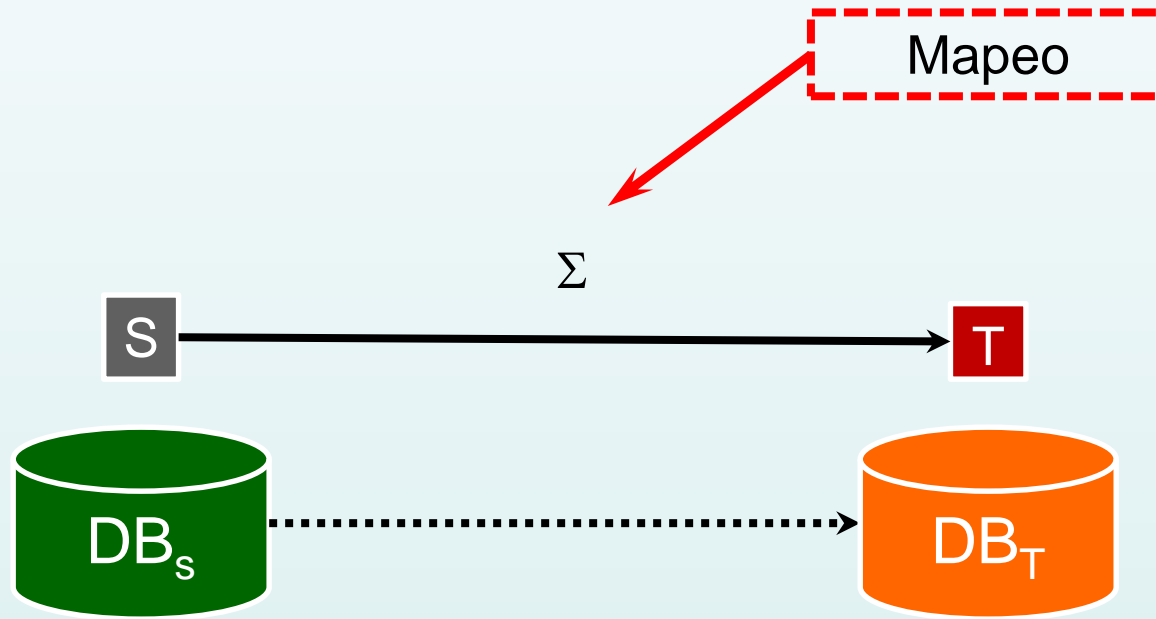
Se consultan datos heterogéneos de diferentes fuentes vía un *esquema virtual global*:



Los datos **NUNCA** están en T ; T convierte la consulta para que las fuentes distintas fuentes la puedan procesar.

Intercambio de datos (materialización)

Se *transforman* datos estructurados bajo un esquema *origen* en datos estructurados bajo un esquema *destino* diferente:



Ejemplos:

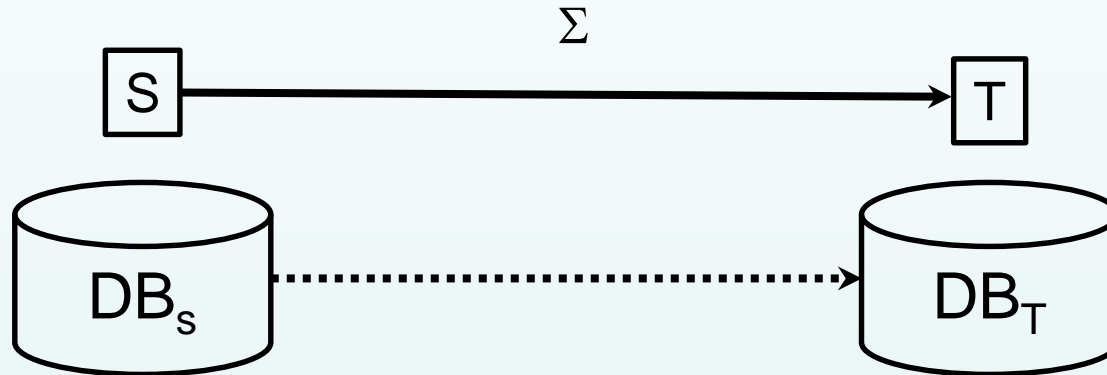
Fusiones,
Adquisiciones,
...

Schema Mappings (Mapeos entre esquemas)

Los *mapeos entre esquema* son las piezas fundamentales para la formalización y estudio de ambos casos:

- Aserciones declarativas de alto nivel que especifican la *relación* entre dos esquemas de BD.
- Hacen que sea posible la separación del *diseño* de la relación entre esquemas y su *implementación*:
 - Son fáciles de generar y manejar (semi-)automáticamente
 - Pueden ser compilados en scripts SQL/XSLT

Mapeos entre esquema



Mapeo $M = (S, T, \Sigma)$

- Esquema *origen* S , esquema *destino* T
- Mapeo Σ

¿Qué constituye un “buen” lenguaje de especificación de mapeos de esquema?

Lenguaje de mapeo entre esquemas

Todo lenguaje de especificación de mapeos debería tener:

- **Copiado** (*Nicknaming*): Copiar cada tabla origen a una tabla destino, y renombrarla.
- **Proyección** (Borrado de columna): Formar una tabla destino borrando una o más columnas de una tabla origen.
- **Incorporación de columnas**: Formar una tabla destino sumando una o más columnas a una tabla origen.
- **Descomposición**: Descomponer una tabla origen en una o más tablas destino.
- **Join**: Formar una tabla destino haciendo join entre dos o más tablas origen.
- **Combinaciones** de estas operaciones.

Lenguaje de mapeo entre esquemas: TGDs!

Veamos cómo éstas se pueden representar en FOL:

- Copiado (*Nicknaming*): $\forall X_1, \dots, X_n (P(X_1, \dots, X_n) \rightarrow R(X_1, \dots, X_n))$
- Proyección (Borrado de columna): $\forall X, Y, Z (P(X, Y, Z) \rightarrow R(X, Y))$
- Agregado de columnas: $\forall X, Y (P(X, Y) \rightarrow \exists Z R(X, Y, Z))$
- Descomposición: $\forall X, Y, Z (P(X, Y, Z) \rightarrow R(X, Y) \wedge T(Y, Z))$
- Join: $\forall X, Y, Z (E(X, Z) \wedge F(Z, Y) \rightarrow R(X, Z, Y))$
- Combinaciones de estas operaciones, por ejemplo: join + agregado + descomposición:

$$\forall X, Y, Z (E(X, Z) \wedge F(Z, Y) \rightarrow \exists W (R(X, Y) \wedge T(X, Y, Z, W)))$$

s-t-TGDs

Para nuestros propósitos, alcanza con una subclase llamada *source-to-target TGDs* (s-t-TGDs), con forma:

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

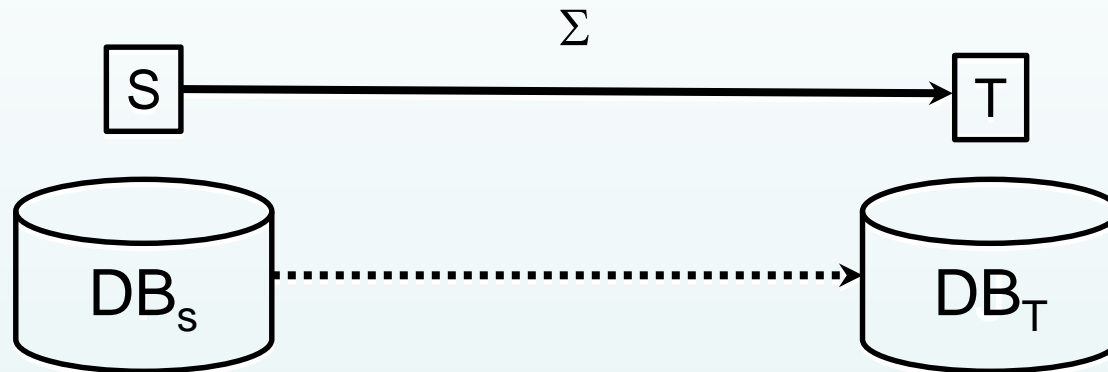
donde ϕ es una conjunción de átomos sobre el esquema *origen* y ψ es una conjunción de átomos sobre el *destino*.

Ejemplo: Esquema destino: relaciones *empleado*, *rol* y *asignación*;
esquema fuente: relaciones *profesor* y *dicta*;

$$\forall PID \ \forall Nom \ \text{profesor}(PID, Nom) \rightarrow \text{empleado}(PID, Nom) \wedge \text{rol}(PID, \text{profesor})$$

$$\forall PID \ \forall Nom \ \forall C \ \text{profesor}(PID, Nom) \wedge \text{dicta}(PID, C) \rightarrow \exists S \ \text{asignacion}(P, C, S)$$

Semántica de los mapeos de esquema



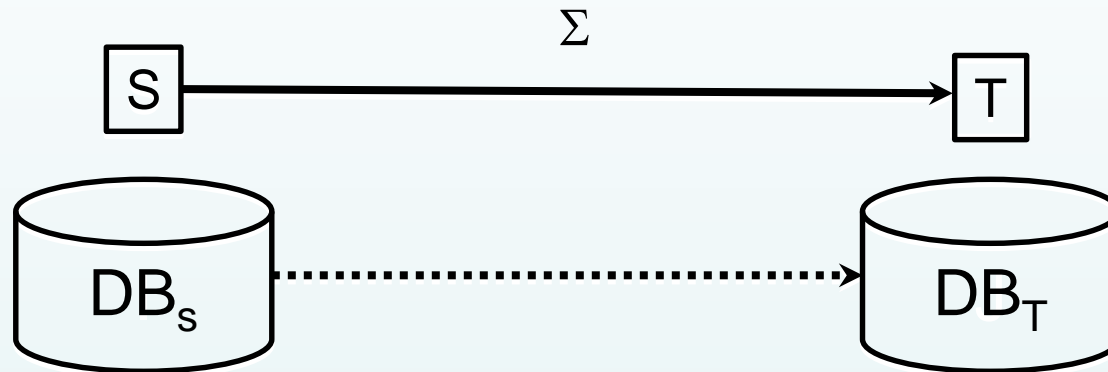
Mapeo $M = (S, T, \Sigma)$, donde Σ es un conjunto de s-t-TGDs.

Desde un punto de vista *semántico*, M puede verse como:

$$Inst(M) = \left\{ (I, J) \mid \begin{array}{l} I \text{ es una instancia sobre el esquema } \textit{origen}, \\ J \text{ es una instancia sobre el esquema } \textit{destino}, \text{ y } (I, J) \models \Sigma \end{array} \right\}$$

Una *solución* para una instancia origen I es una instancia destino J tal que $(I, J) \in Inst(M)$ (i.e., $(I, J) \models \Sigma$).

Semántica de los mapeos entre esquemas



Mapeo $M = (S, T, \Sigma)$, donde Σ es un conjunto de s-t-TGDs.

Informalmente, esta notación significa que J contiene todas las tuplas que surgen de aplicar las restricciones en Σ sobre tuplas en I .

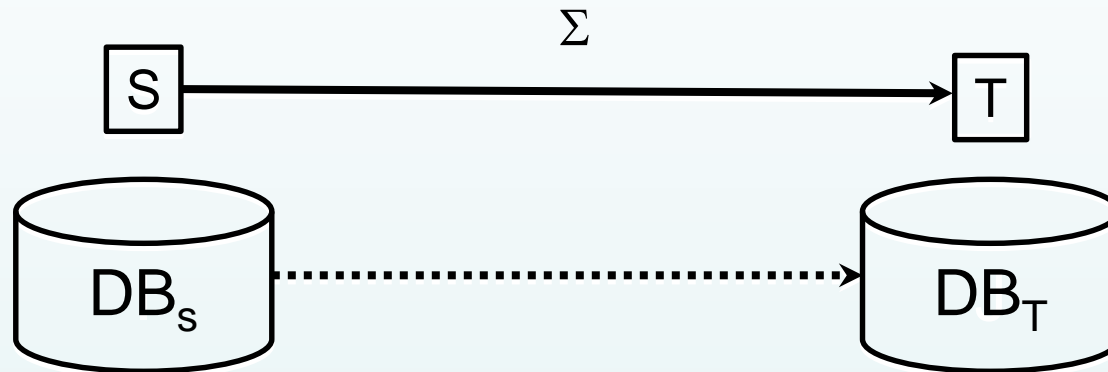
Formalmente: $(I, J) \models \Sigma$

uede verse como:

una *origen*,
una *destino*, y $(I, J) \models \Sigma$

Una **solución** para una instancia origen I es una instancia destino J tal que $(I, J) \in \text{Inst}(M)$ (i.e., $(I, J) \models \Sigma$).

Intercambio de datos



- El problema computacional del **intercambio de datos** a través del mapeo $M = (S, T, \Sigma)$ es entonces: *dada la instancia I , **construir una solución** J para I .*
- La **dificultad** de este problema radica en que:
 - Típicamente hay **muchas** soluciones posibles
 - ¿Cómo decidir *cuál de todas es la mejor*?

El Chase

- Teorema: Sea $M = (S, T, \Sigma)$ un mapeo, Σ es un conjunto de s-t-TGDs). Entonces, para cada instancia origen I :
 - El procedimiento *chase* produce una *solución universal*, denotada con $chase_M(S, \Sigma)$.
 - El tiempo de ejecución para el chase está acotado por un *polinomio* en el tamaño de I (complejidad de datos).
- El chase puede ser *aumentado* también con dependencias sólo target (t-TGDs).

Una solución más general: OBDA

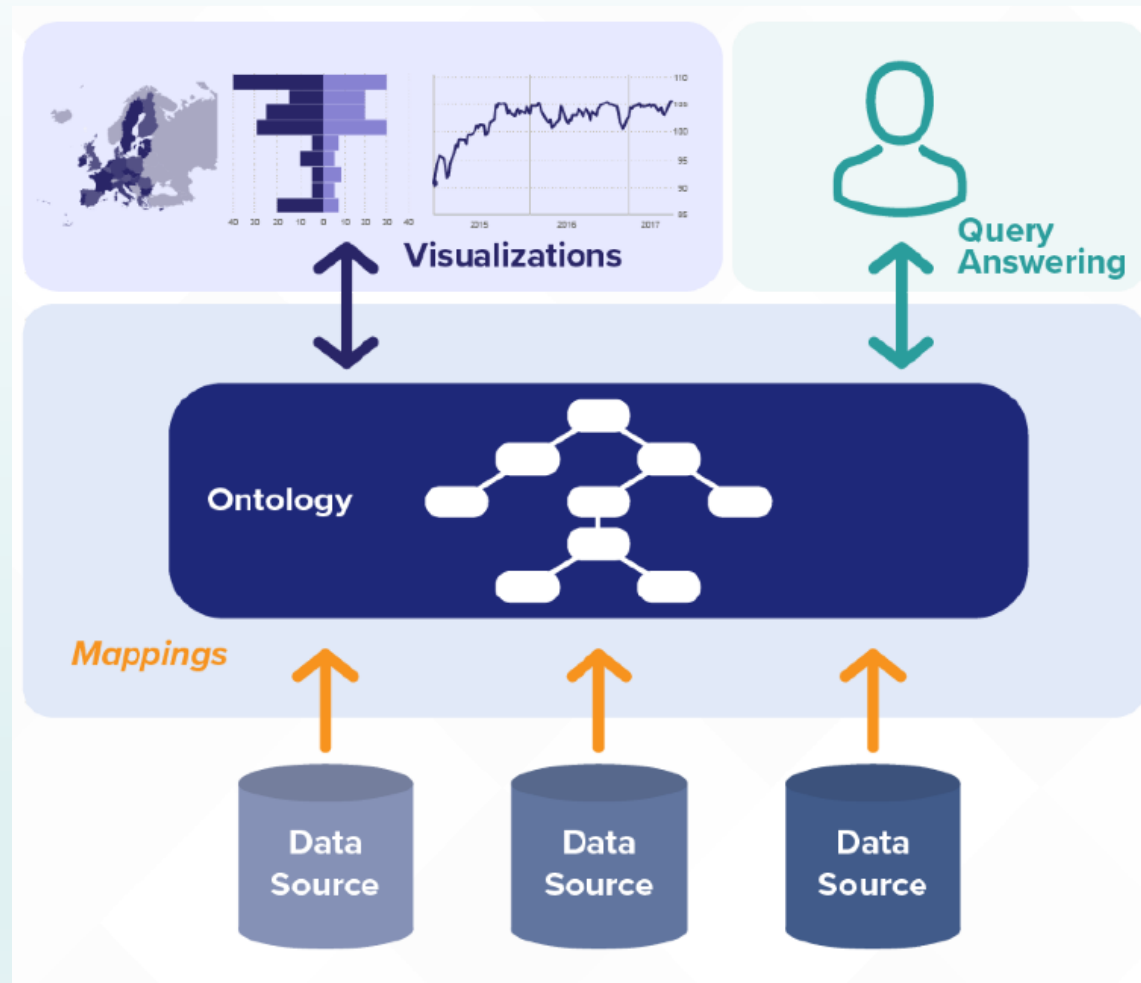
Administrar los datos adoptando **principios** y **técnicas** estudiados en el área de **KR&R**:

- **Representación conceptual de alto nivel** del dominio en términos de una **ontología**: los usuarios hacen sus queries en este lenguaje (+amigable, en términos del dominio de aplicación).
- No se necesita mover/cambiar/arreglar los datos fuentes.
- **Mapear** la ontología a las fuentes de datos (distintos niveles de reglas de traducción).
- **Automáticamente** se **traducen** los requerimientos a **consultas** a las fuentes de datos y viceversa.

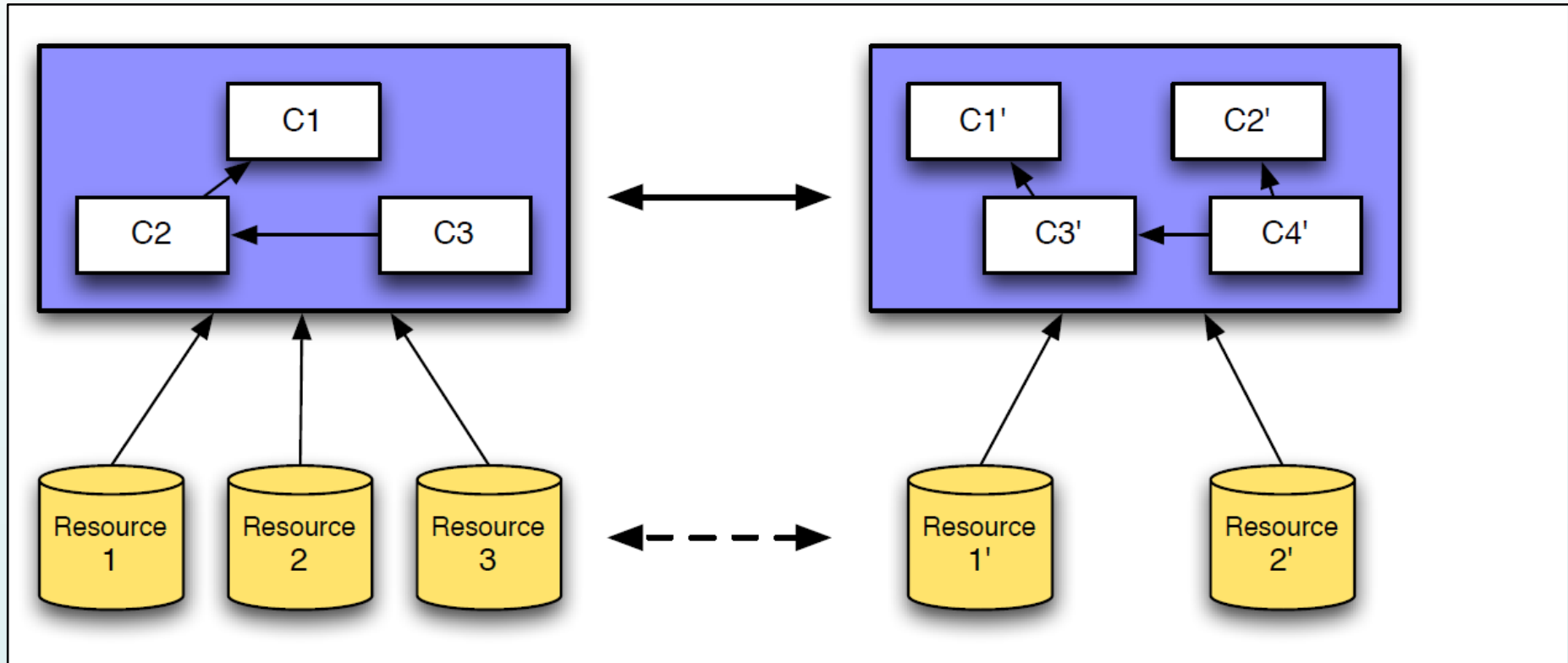
Ontology-Based Data Access (OBDA)

- Meta: alcanzar *transparencia* lógica en el acceso a los datos:
 - Esconder dónde y cómo están *almacenados* los datos.
 - Presentar al usuario una *vista conceptual* de los datos.
 - Usar un formalismo *semánticamente rico* para la vista conceptual.

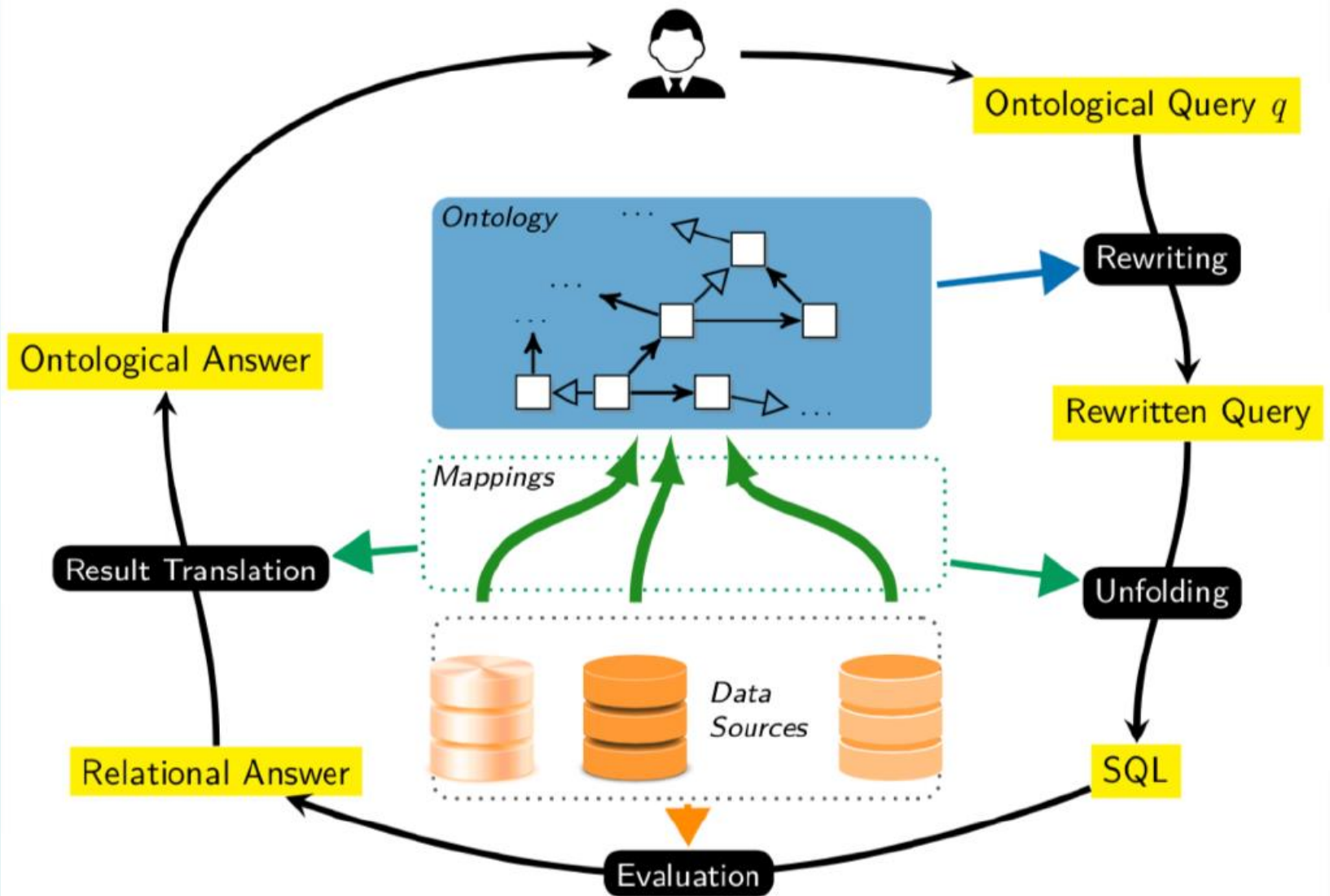
Ontology-Based Data Access (OBDA)



Ontology-Based Data Access (OBDA)



Ontology-Mediated Query Answering (OMQA)



Ejercicios Tema 4

1. Definir los siguientes conceptos usando los componentes y constructores de DLs:
 - “abuela materna” , “tío sin hijos”, “sobrino de una nuera”, “padre con al menos 3 hijos, dos de los cuales es una mujer”.

Puede usar conceptos intermedios pero debe definirlos en términos de los conceptos definidos en la slide 28.

2. Dado el programa Datalog anterior, indique si:

- Se puede derivar $parent(evan, david)$?
- Se puede derivar $ancestor(carla, david)$?

En el caso donde se pueda muestre el árbol de derivación.

3. Dada la consulta conjuntiva:

$$Q(X,Y) = \exists Z. parent(X,Y) \wedge parent(Z,Y)$$

¿Qué regla debemos agregarle a P (slide 49)? ¿Cual sería el conjunto de respuesta? Verifiquelo en ABCDatalog.



Ejercicios Tema 4

4. ¿Cómo sería la regla que hay que agregar en P (slide 49) dada la consulta Booleana:
 $Q1() = \exists Z. parent(X,Y) \wedge parent(Z,Y)?$
5. Escriba un programa Datalog para resolver el problema del mundo de bloques planteado en el práctico 3. Generalice su programa de manera que se pueda hacer las siguientes consultas: para cualquier bloque X, Y, “está el bloque X sobre la mesa”?, “está X apilado sobre Y”?, “está X debajo de Y?”, “esta X inmediatamente sobre Y”?, “esta X inmediatamente por debajo de Y”? Pruebe la correctitud del programa utilizando la herramienta ABCDatalog.
6. Proponga un ejemplo de dos esquemas de bases de datos (fuente y destino), muestre las s-t-TGDs necesarias para pasar de uno a otro, similar a slide 91, explique la transformación necesaria. Defina una instancia de bases de datos en el esquema fuente y muestre cual sería la solución (la instancia destino) para el problema de intercambio de datos.

Referencias

- Computational Intelligence: A Logical Approach - D.Poole, A. Mackworth, R. Goebel Oxford University Press.
- [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).
- “*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.
- Phokion G. Kolaitis: “*A Tutorial on Schema Mappings and Data Exchange*”, dictado en DEIS 2010, Alemania, noviembre de 2010.