

Bases de Datos

Clase 12: Ontology-based Data Access

Vanina Martinez

Bases de Datos: 2do Cuatrimestre 2019

Conocimiento ontológico:

Inteligencia Artificial

Representación de Conocimiento y

Razonamiento

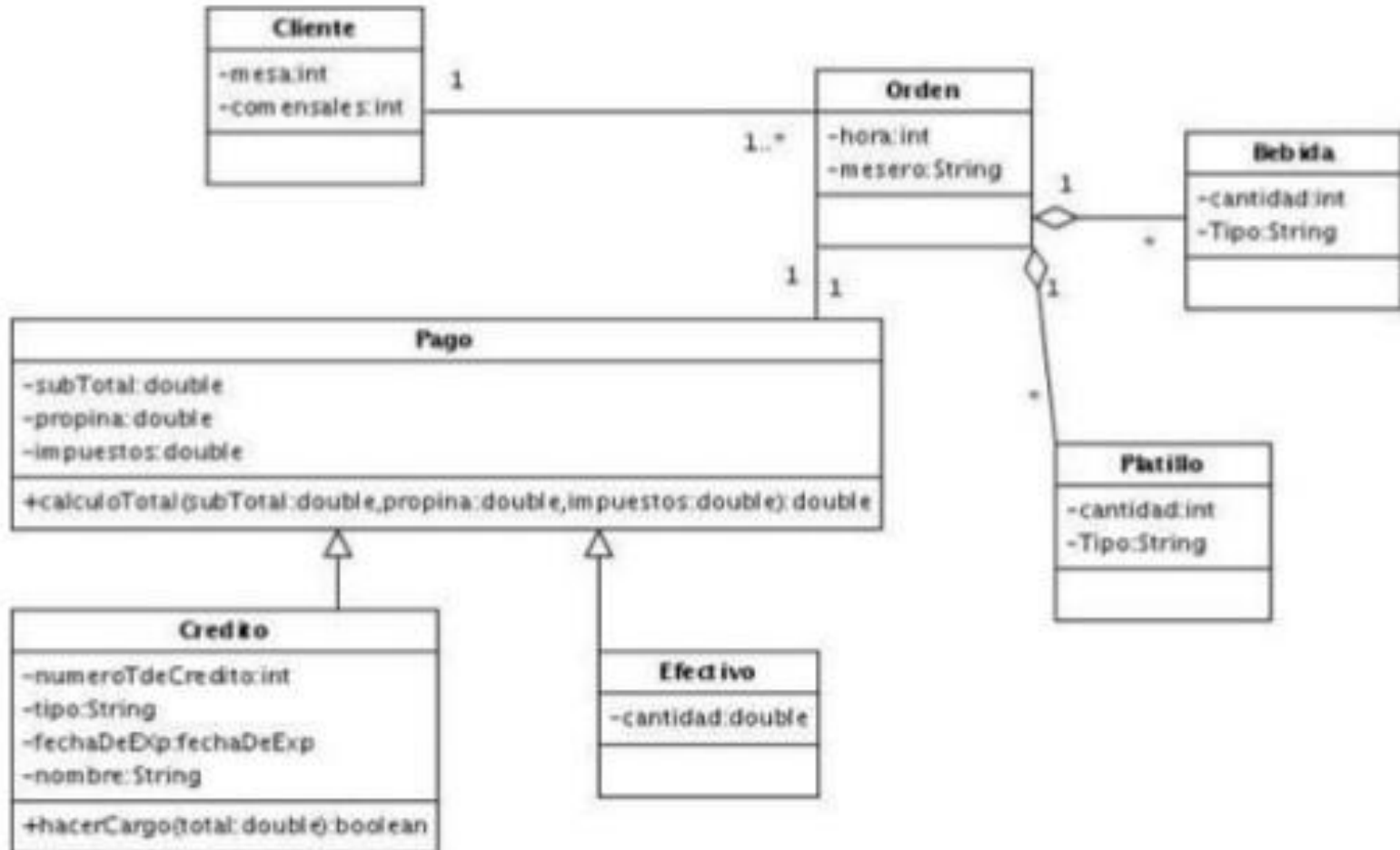
Conocimiento ontológico

- Una conceptualización es una *vista* abstracta y simplificada del mundo que queremos representar.
- *Ontología*: es un esquema de representación que describe una conceptualización formal de un dominio de interés.
- Usualmente es una teoría *lógica* que expresa la conceptualización explícitamente en un lenguaje (declarativo).
- Facilita el *reúso* e *intercambio* de conocimiento.
- Define el vocabulario con el cual las consultas y aserciones se intercambian entre agentes.

Conocimiento ontológico

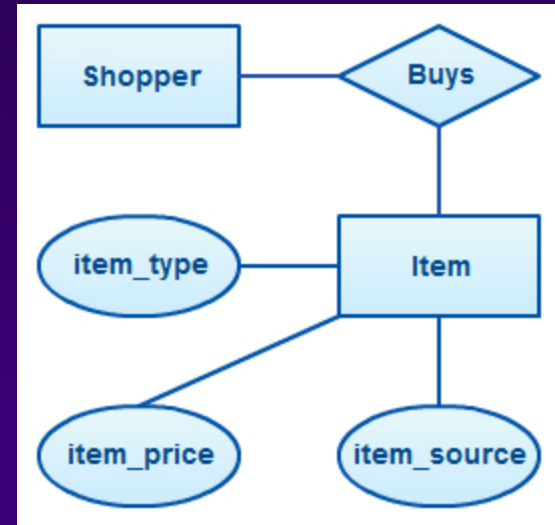
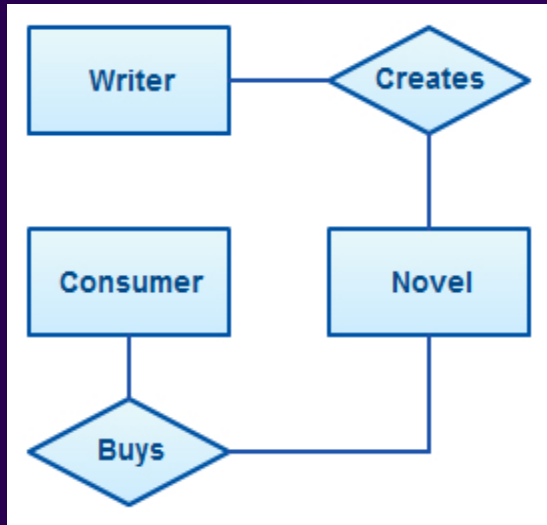
- Cada *base de conocimiento* o sistema basado en conocimiento está asociado a una conceptualización (explícita o implícitamente).
 - Para estos sistemas, lo que *existe* es lo que se *representa* (*aunque puede representarse la incertidumbre y la incompletitud*).
- Cuando el conocimiento de un dominio se representa en un formalismo declarativo, el conjunto de objetos que puede representarse se llama el *universo de discurso*.
 - Este conjunto de objetos y las relaciones describibles entre ellos se reflejan en el *vocabulario* de representación con el cual el programa basado en conocimiento representa el conocimiento.

Ejemplo Conceptualización



Fuente Imagen: <https://es.slideshare.net/nedowwhaw/diagrama-de-clases-16208245>

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>

Conocimiento ontológico

- En el contexto de *Inteligencia Artificial* (IA), o *Representación de Conocimiento y Razonamiento* (KR&R), podemos describir la ontología de un programa definiendo un conjunto de *términos representacionales*.
 - Las definiciones de nombres de *entidades* en el universo de discurso (clases, relaciones, funciones, y otros objetos) se asocian con descripciones de lo que los nombres significan y axiomas formales que restringen la interpretación y uso *bien formado* de esos términos.
 - Formalmente, se puede decir que una ontología *describe* una *teoría lógica*.

Representación de conocimiento

- KR&R se enfoca en métodos para proveer descripciones de alto nivel del mundo, que pueden usarse para construir aplicaciones inteligentes.
 - Aquí, “*inteligencia*” es la habilidad de un sistema de encontrar consecuencias *implícitas* de conocimiento explícito.
- Los enfoques de KR se dividen en dos categorías:
 - *Formalismos basados en lógica*: evolucionaron de la intuición de que el cálculo de predicados se puede usar para capturar hechos sobre el mundo de manera no ambigua.
 - *Representaciones no basadas en lógica*: inspiradas en nociones más bien cognitivas; por ejemplo, estructuras de redes, y representaciones basadas en reglas derivadas de experimentos sobre actividades humanas.

Representación de conocimiento

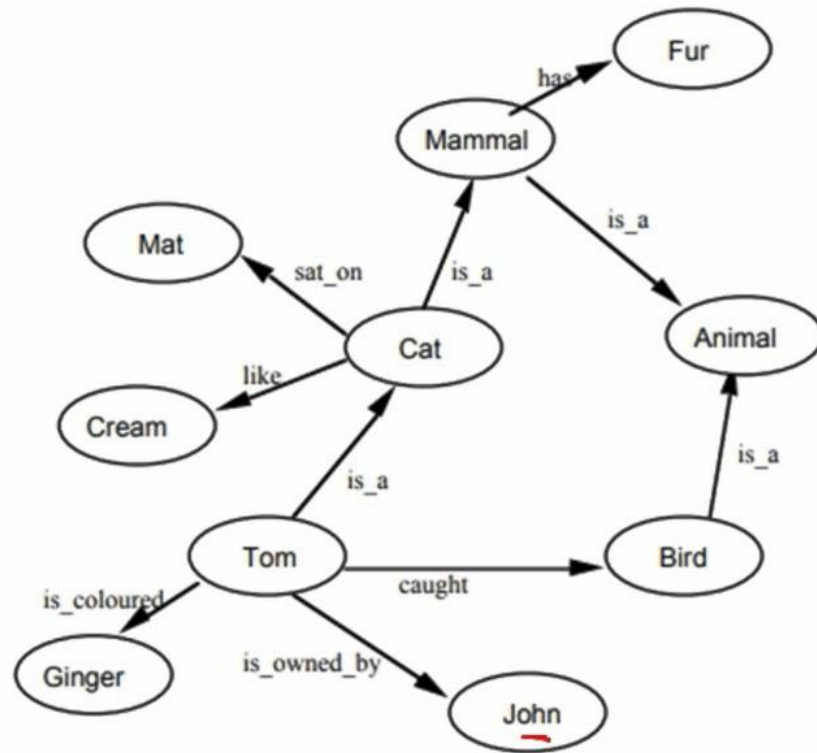
- En el enfoque basado en *lógica*, el lenguaje de representación es en general una variante del cálculo de predicados de primer orden y la tarea de razonar implica la verificación de *consecuencias* lógicas.
- Los enfoques *no lógicos*, se basan en interfaces gráficas y el conocimiento es representado 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 semantica 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.
- Por esto, cada sistema se comportaba de manera diferente a los otros, aun cuando los componentes se veían muy parecidos o con nombres de relaciones idénticas.
- 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.
- Las características de cada DL están definidas por los *constructores* que establecen relaciones entre objetos.

DLs: Lenguaje de descripción

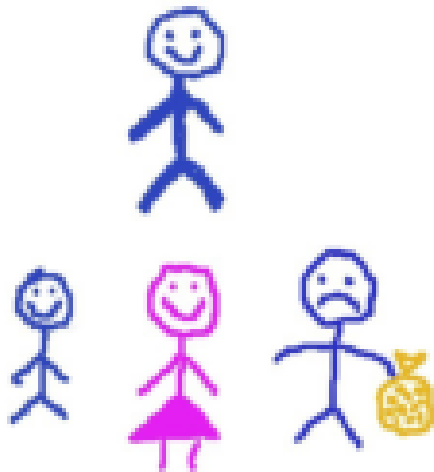
- Un lenguaje de descripción se caracteriza por un conjunto de *constructores* para crear conceptos complejos y roles a partir de los básicos:
 - Los conceptos corresponden a *clases*: interpretados como un conjunto de objetos.
 - Roles corresponden a *relaciones*: interpretados como relaciones binarias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

DLs: Semántica (FOL)

- 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.

Ejemplo DLs

El padre contento:



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

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.

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) $\Rightarrow C \sqsubseteq \emptyset$?

Otro ejemplo:

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

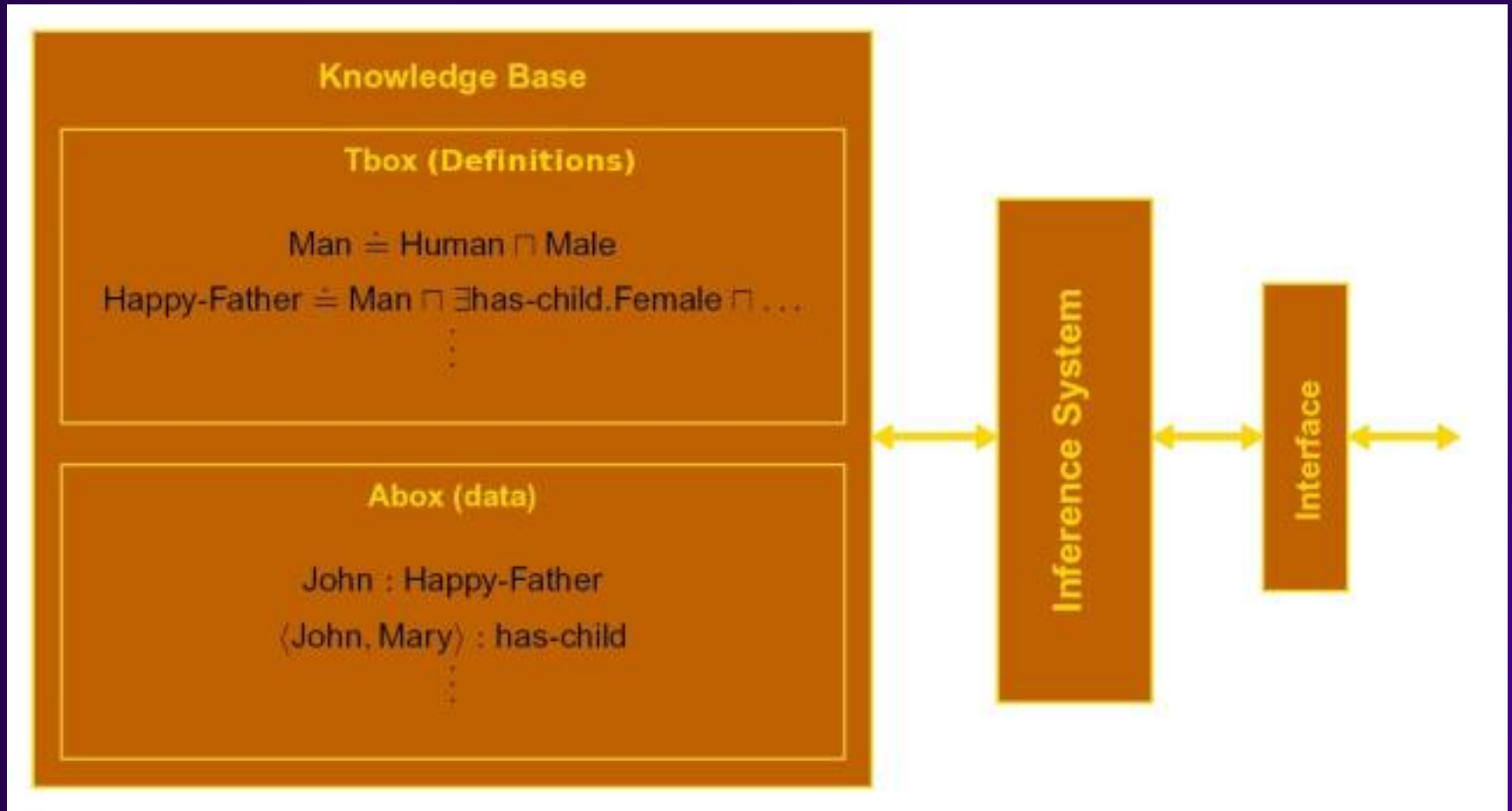
La A-Box:

alicia:Madre

(alicia,betty):hijo-de

(alicia,carlos):hijo-de

Arquitectura para DLs



Áreas de aplicación

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.

Lingüística Computacional

- Inferencia y '**background knowledge**': resolución de referencias, etc.

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).

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.
- Como fuente de definiciones precisas de términos que pueden ser compartidas entre aplicaciones.
- Cuanto mayor sea el grado de formalización y mayor regularidad, más fácil va a ser que la ontología sea manipulable automáticamente.

Conocimiento ontológico:

Bases de Datos

Inteligencia Artificial

Representación de Conocimiento y

Razonamiento

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.

¿Solución?

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

- Proveer una **representación conceptual de alto nivel** del dominio en términos de una **ontología**.
- No se necesita mover los datos.
- **Mapear** la ontología a las fuentes de datos.
- Todos los **requerimientos de información** se hacen en términos de la ontología.
- Se usan servicios de **inferencia automáticos** que **traducen** los requerimientos en **consultas** a las fuentes de datos.

¿Solución?

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

- Proveer una **representación conceptual de alto nivel** del dominio en términos de una **ontología**.
- No se necesita mover los datos.
- **Mapear** la ontología a las fuentes de datos.
- Todos los **requerimientos** de **información** se hacen en términos de la ontología.
- Se **traducen** automáticamente las **consultas** a las distintas fuentes de datos.

Ontology-Based Data Access (OBDA)

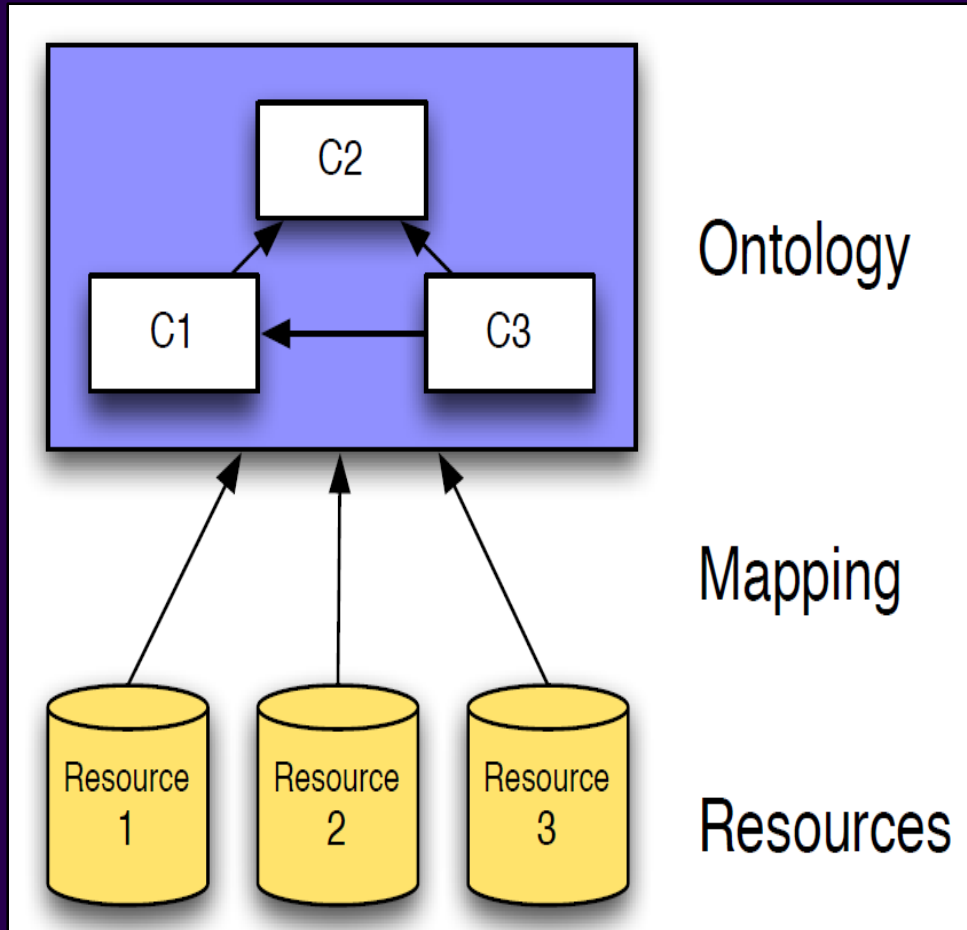
- Meta: alcanzar *transparencia* lógica en el acceso a los datos:
 - Esconder dónde y cómo están *almacenados* los datos.
 - Presentar al usuario una *vista conceptual* de los datos.
 - Usar un formalismo *semánticamente rico* para la vista conceptual.
- Objetivo similar al de *integración de datos*, pero con una descripción conceptual rica de la vista global.

¿Solución?

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

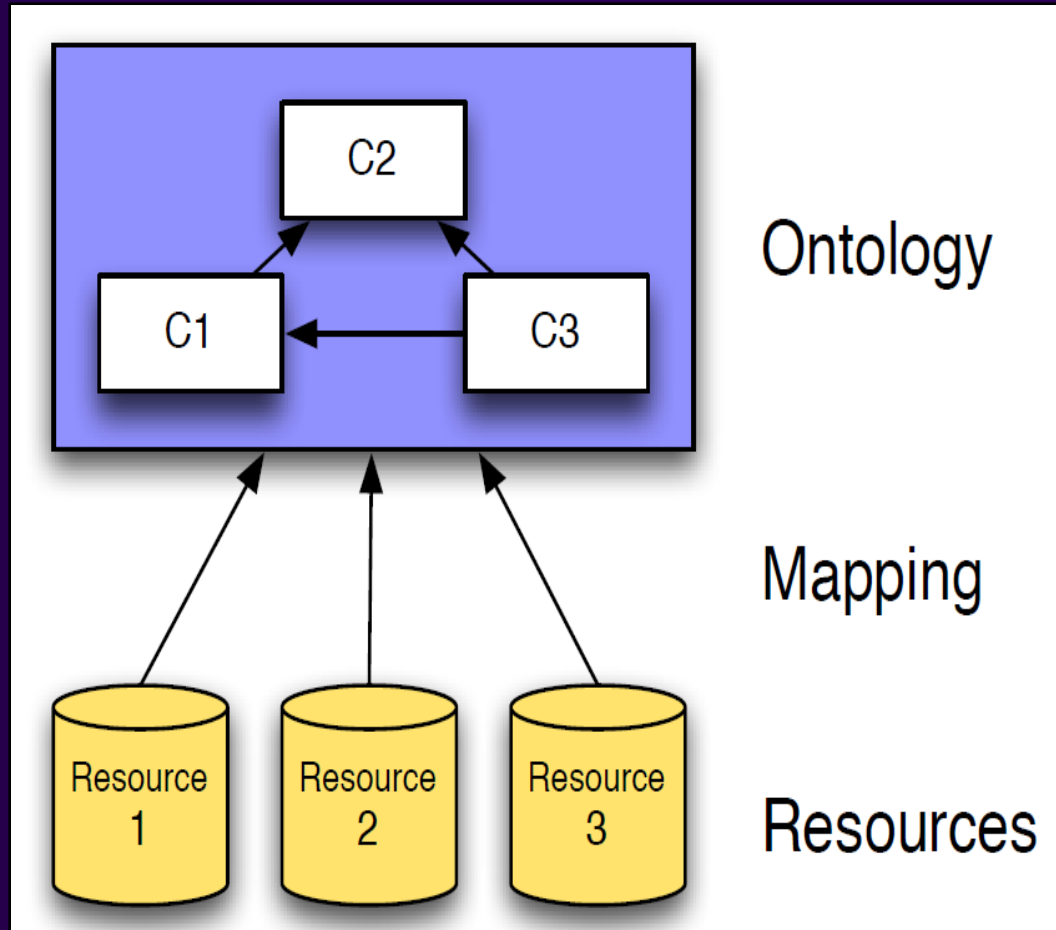
- Proveer una **representación conceptual de alto nivel** del dominio en términos de una **ontología**.
- No se necesita mover los datos (definir un lenguaje **Target**).
- **Mapear** la ontología a las fuentes de datos (**Mapeos de esquemas**).
- Todos los **requerimientos de información** se hacen en términos de la ontología (lenguaje **Target**).
- Se **traducen** automáticamente las **consultas** a las distintas fuentes de datos.

Ontology-Based Data Access (OBDA)



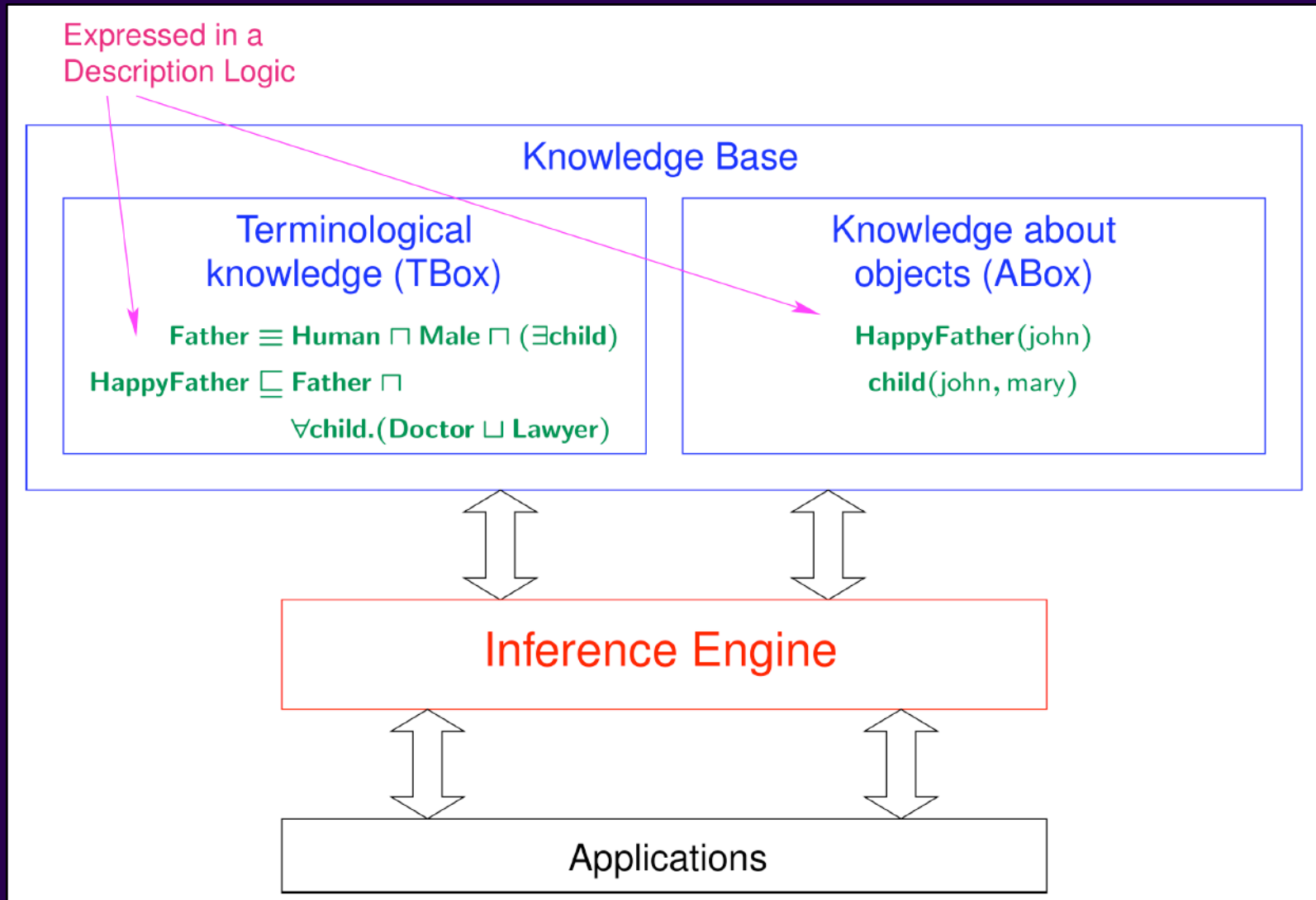
- La ontología provee un vocabulario global y se usa como una vista conceptual.
- Los mapeos enlazan semánticamente las fuentes y la ontología.
- Las fuentes de datos son externas y heterogéneas.

Ontology-Based Data Access (OBDA)



- La ontología provee un vocabulario global y se usa como una vista conceptual.
- Los mapeos enlazan semánticamente las fuentes y la ontología.
- Las fuentes de datos son externas y heterogéneas.

Arquitectura de un sistema OBDA (con DLs)



Ontology-Based Data Access (OBDA)

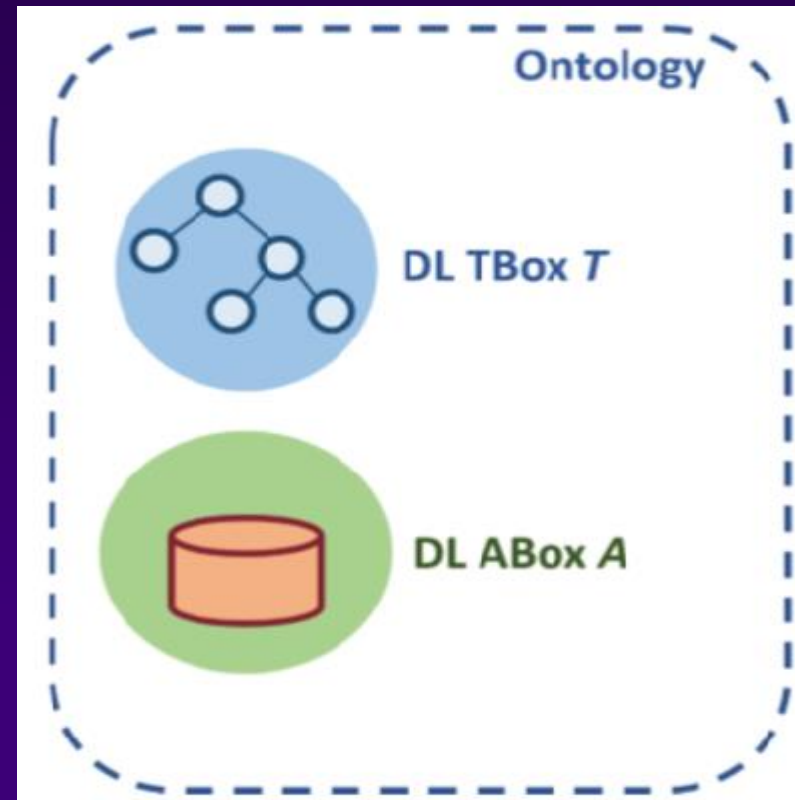
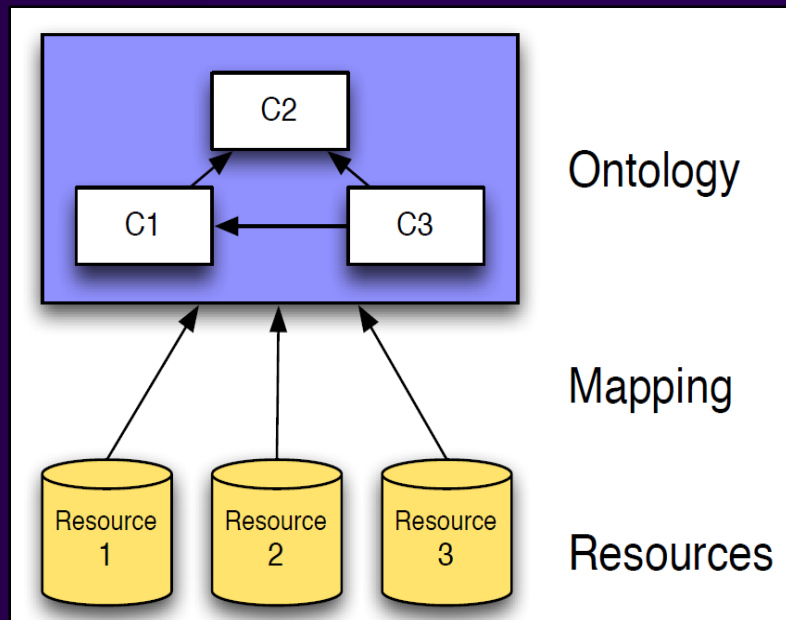
- Meta: alcanzar transparencia lógica en el acceso a los datos.
 - Esconder dónde y cómo están almacenados los datos.
 - Presentar al usuario una vista conceptual de los datos.
 - Usar un formalismo semánticamente rico para la vista conceptual.
- Formalización de:
 - Lenguajes
 - Metodologías
 - Herramientas

Para *especificar*, *construir*, y *administrar* ontologías que se usan en sistemas de información.

Diseño de un framework OBDA

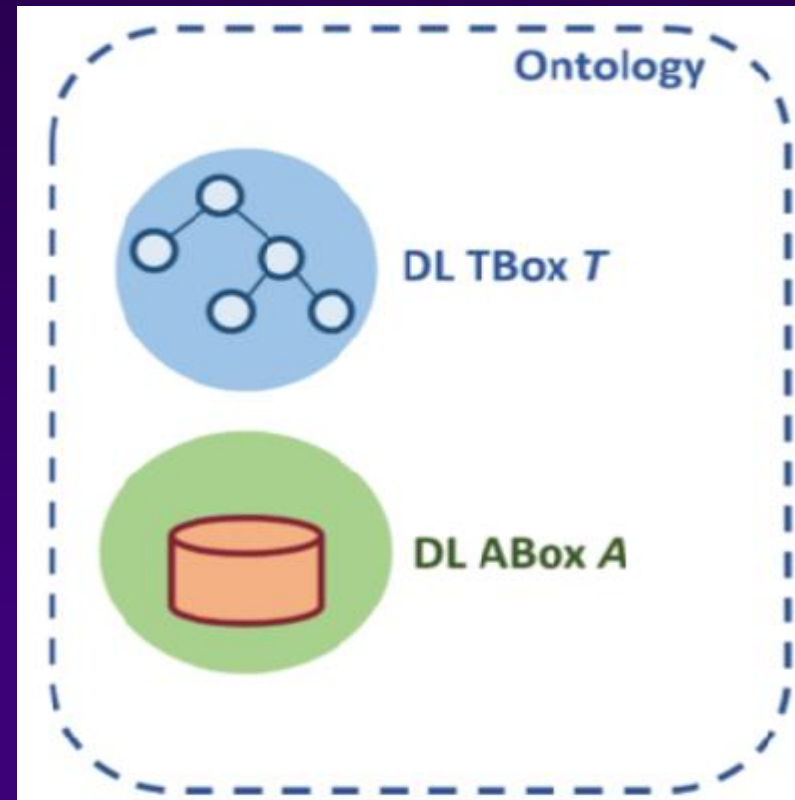
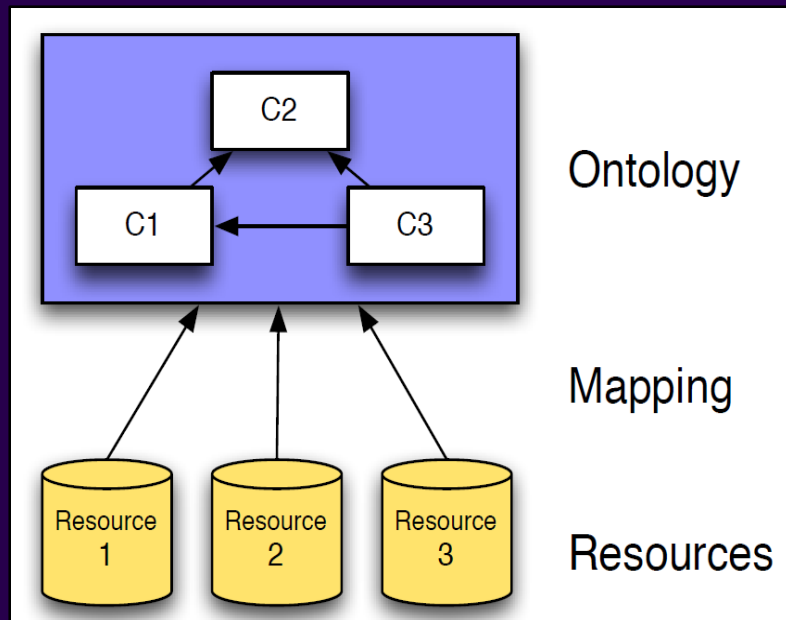
- ¿Cuál es el **lenguaje de ontología “correcto”**?
- ¿Cuál es el **lenguaje de mapeo “correcto”**?
- ¿Cuál es el **lenguaje de consulta “correcto”**?
- **Compromiso** entre el **poder expresivo**, la **facilidad para definir expresiones significativas en el lenguaje (user friendly)** y la **eficiencia computacional** al responder las consultas!
- Si queremos acceder a “Big Data” la **eficiencia** con respecto a los datos **es crucial**.

Ontology-based Query Answering



La ABox puede verse como una instancia de bases de datos con los mismos predicados que la ontología y que contenga tanto objetos como valores.

Ontology-based Query Answering



La TBox representa la vista conceptual del dominio.

Diseño de un framework OBDA

- Cúal es el **lenguaje de ontología “correcto”**?
 - Lenguaje gráfico de alto nivel
 - DLs
 - Datalog+/- (Reglas lógicas con existenciales - TGDs)

Ontology-Based Data Access (OBDA)

- Una Lógica de Descripción se caracteriza por:
 - Un lenguaje de descripción: cómo formar conceptos y roles.

$$Father \equiv Human \sqcap Male \sqcap \exists hasChild$$

- Un mecanismo para especificar conocimiento acerca de los conceptos y roles (una *TBox* o *axiomas terminológicos*).

$$T = \{ Father \equiv Human \sqcap Male \sqcap \exists hasChild,$$

$$HappyFather \sqsubseteq Father \sqcap \forall hasChild. (Doctor \sqcup Lawyer) \}$$

- Un lenguaje de consulta: $\{ Human \sqcap Male \sqcap \exists hasChild \sqcap \forall hasChild. (Doctor), Human \sqcap Male \sqcap \exists hasChild \sqcap \forall hasChild. (Layer) \}$

Ontology-Based Data Access (OBDA)

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

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

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

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

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

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:

$$cientifico \sqsubseteq \exists esAutor de$$

- 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 $sen-emp \times emp \sqsubseteq moreThan$	$sen-emp(X), emp(Y) \rightarrow moreThan(X, Y)$
(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 $func(reports)$	$reports(X, Y), reports(X, Z) \rightarrow Y = Z$

Lenguajes de consulta: opciones

- Usar el lenguaje de ontología:
 - Los lenguajes de ontologías están diseñados para capturar relaciones intencionales. Son pobres en expresividad.
- Full SQL (o FOL):
 - Problema: en la presencia de información incompleta, query answering es indecidible (validez FOL).

A buen trade-off es usar consultas conjuntivas (CQs) o union de CQs (UCQs), que corresponde a SQL/algebra relacional (union) consultas select-project-join.

Formalismos basados en Datalog

DLs
(DL-Lite, EL,...)

**Restricciones
relacionales**
(IDs, FKDs)

Datalog

Sin perder tratabilidad...

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).
- Se extiende Datalog con *dependencias* y restricciones de *integridad*... pero con *limitaciones sintácticas* sobre las reglas.
- Distintas restricciones en las TGDs dan lugar a *diferentes lenguajes* con distinto poder expresivo y complejidad computacional (para tareas como *query answering*).

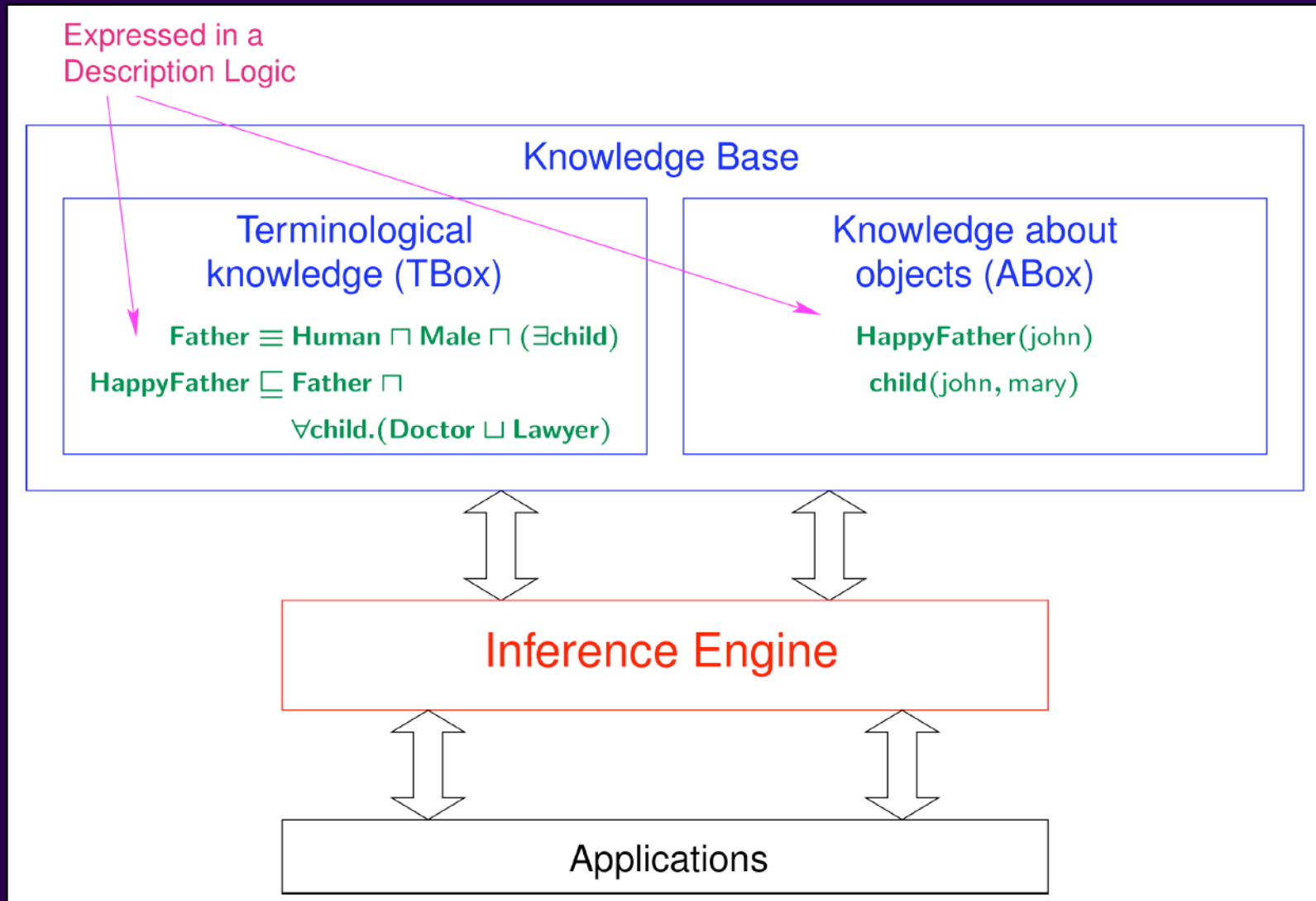
Semántica de Respuestas a Consultas

- 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 $\mu: \Delta \cup \Delta_N \cup \mathcal{V} \rightarrow \Delta \cup \Delta_N \cup \mathcal{V}$:
 - 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** $Q(D)$: 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 $\mu(\mathbf{X}) = t$.

Semántica de Respuestas a Consultas

- **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 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



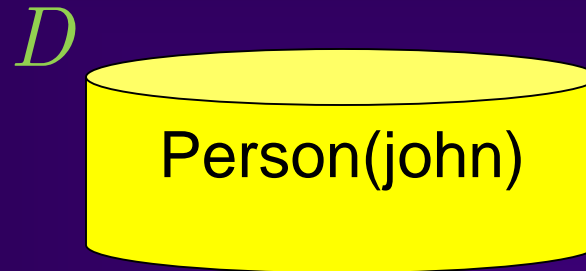
Chase

- 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 Σ

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



Σ

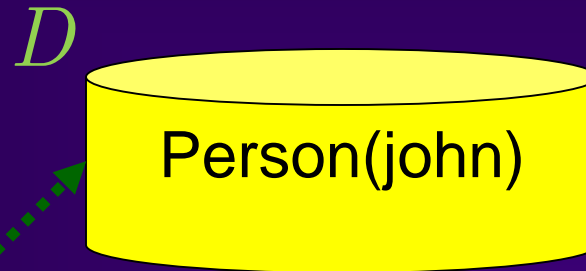
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

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

Chase

Input. Base de datos D , conjunto de TGDs Σ

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



Σ

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

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

Chase

Input. Base de datos D , conjunto de TGDs Σ

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



Σ

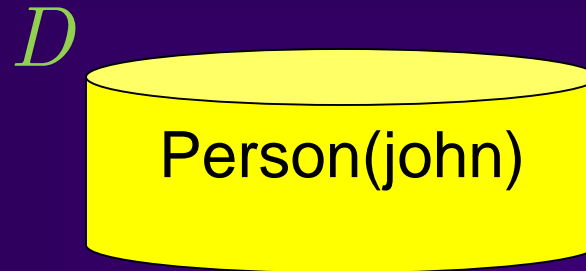
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

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

Chase

Input. Base de datos D , conjunto de TGDs Σ

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



Σ

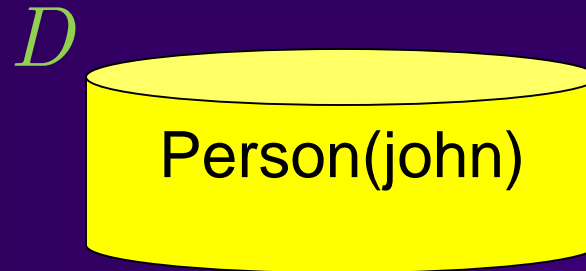
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

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

Chase

Input. Base de datos D , conjunto de TGDs Σ

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



Σ

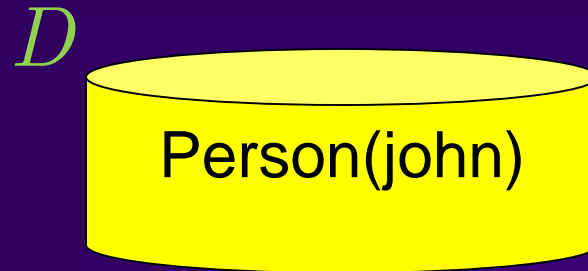
$\forall P \text{ person}(P) \rightarrow \exists F \text{ father}(F,P) \quad \forall F \forall P \text{ father}(F,P) \rightarrow \text{person}(F)$

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

Chase

Input. Base de datos D , conjunto de TGDs Σ

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

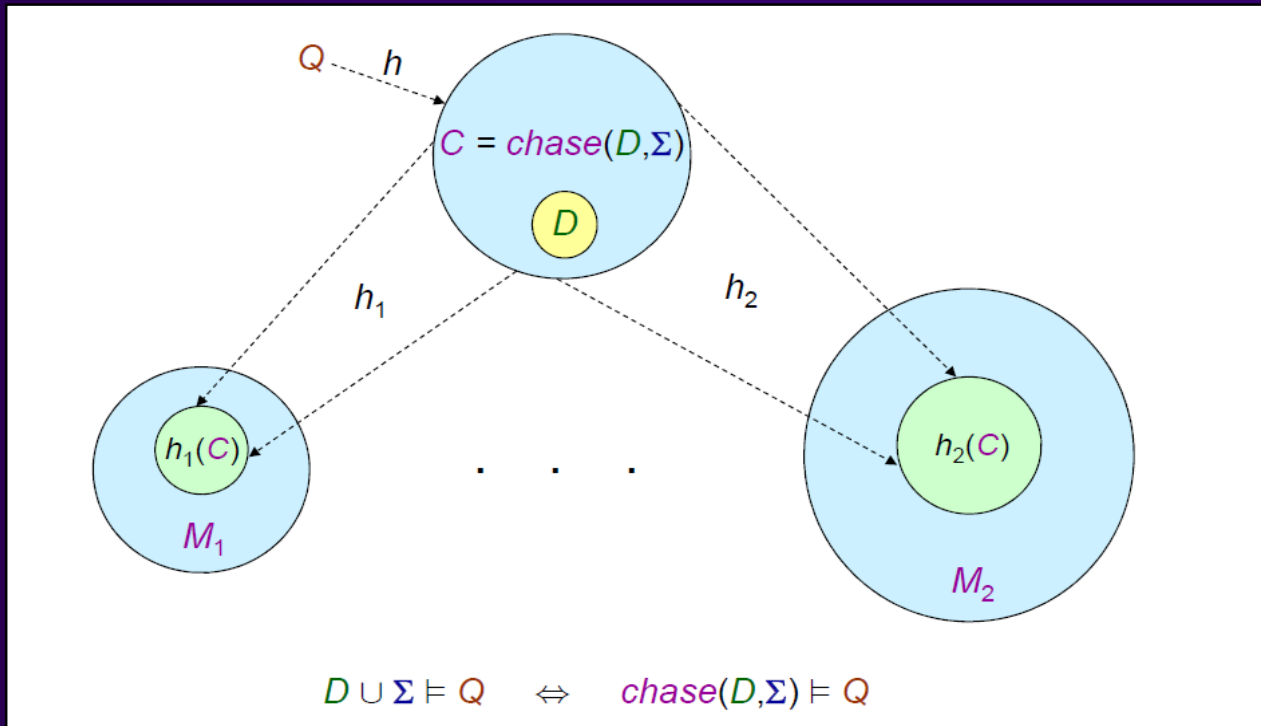


$$\text{chase}(D, \Sigma) = D \cup \{father(z_1, john), person(z_1), father(z_2, z_1), \dots\}$$

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} .
- Se asume un conjunto de EGDs **separables**; intuitivamente significa que las EGDs y TGDs son independientes entre sí.

Ejemplo

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

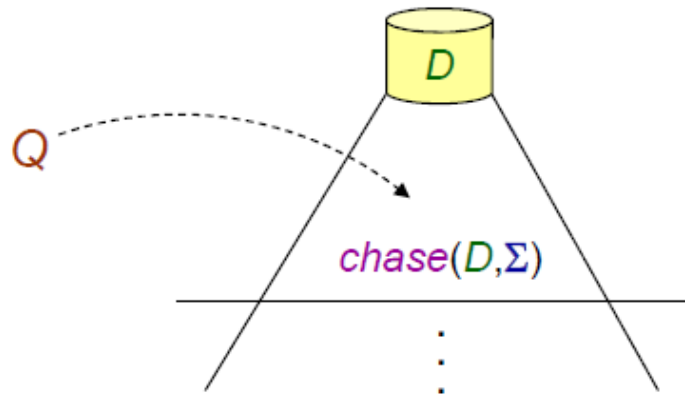
$$\Sigma_T = \{ \text{works_in}(X, D) \rightarrow \text{emp}(X), \\ \text{manager}(X) \rightarrow \exists Y \text{supervises}(X, Y), \\ \text{supervises}(X, Y) \wedge \text{directs}(X, D) \rightarrow \text{works_in}(Y, D) \}$$

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

Resultados positivos

Query Answering con **Inclusion Dependencies** es **decidable**

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



[Johnson & Klug, JCSS 84]

Full TGDs (Datalog)

- Una TGD se dice **full** si todas las variables en la regla están cuantificadas universalmente.


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

- La recursividad permite expresar consultas AR y CRT no pueden.
- No tiene negación \Rightarrow permite expresar un subconjunto de AR y CRT.
- QA es decidable: para queries conjuntivas está en **L-TIME** si la cabeza es vacía, **NLTIME-completo** para reglas **full lineal**, y **PTIME-completo** para **full TGDs** (en complejidad *data*).

Guarded TGDs

- 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** (se parece bastante a un árbol) \Rightarrow query answering decidable
- Query answering es **PTIME-completo** en complejidad *data*.
- Extiende la Lógica de descripción **ELH** (misma complejidad *data*).

Guarded TGDs


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

EL TBox	Traducción a TGDs
$A \sqsubseteq B$	$\forall X A(X) \rightarrow B(X)$
$A \sqcap B \sqsubseteq C$	$\forall X A(X), B(X) \rightarrow C(X)$
$\exists R.A \sqsubseteq B$	$\forall X R(X, Y), A(Y) \rightarrow B(X)$
$A \sqsubseteq \exists R.B$	$\forall X A(X) \rightarrow \exists Y R(X, Y), B(Y)$
$R \sqsubseteq P$	$\forall X \forall Y R(X, Y) \rightarrow P(X, Y)$

TGDs Lineales

- 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 rewritability*).
- Extiende la (familia de) lógicas de descripción *DL-Lite* (misma complejidad data).

TGDs Lineales

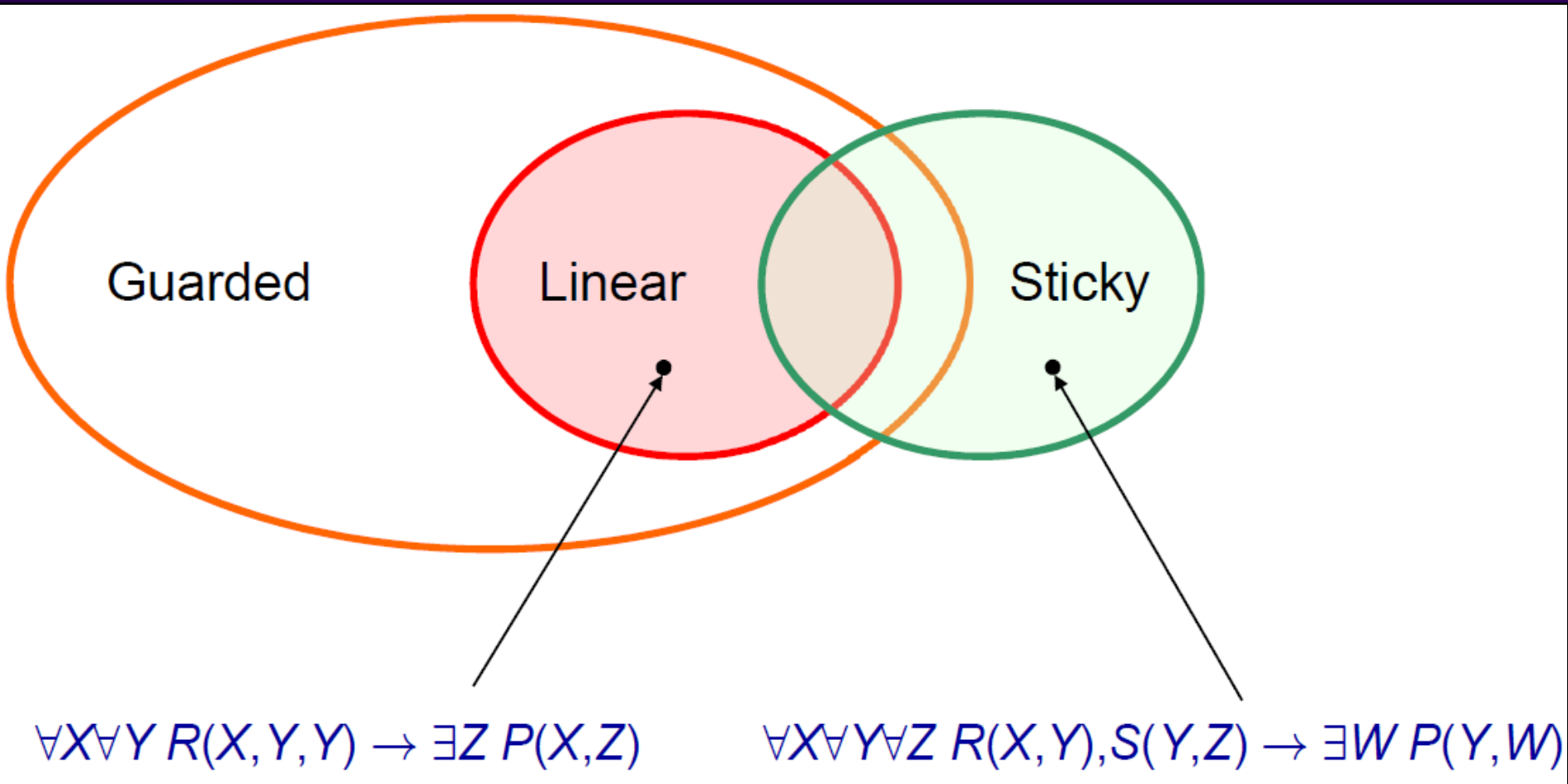
- **DL-Lite** familia de lógicas de descripción con data complejidad AC_0 (OWL 2 QL).

DL-Lite TBox	Traducción a TGDs
$A \sqsubseteq B$	$\forall X A(X) \rightarrow B(X)$
$A \sqsubseteq \exists R$	$\forall X A(X) \rightarrow \exists Y R(X, Y)$
$\exists R \sqsubseteq A$	$\forall X \forall Y R(X, Y) \rightarrow A(X)$
$R \sqsubseteq P$	$\forall X \forall Y R(X, Y) \rightarrow P(X, Y)$

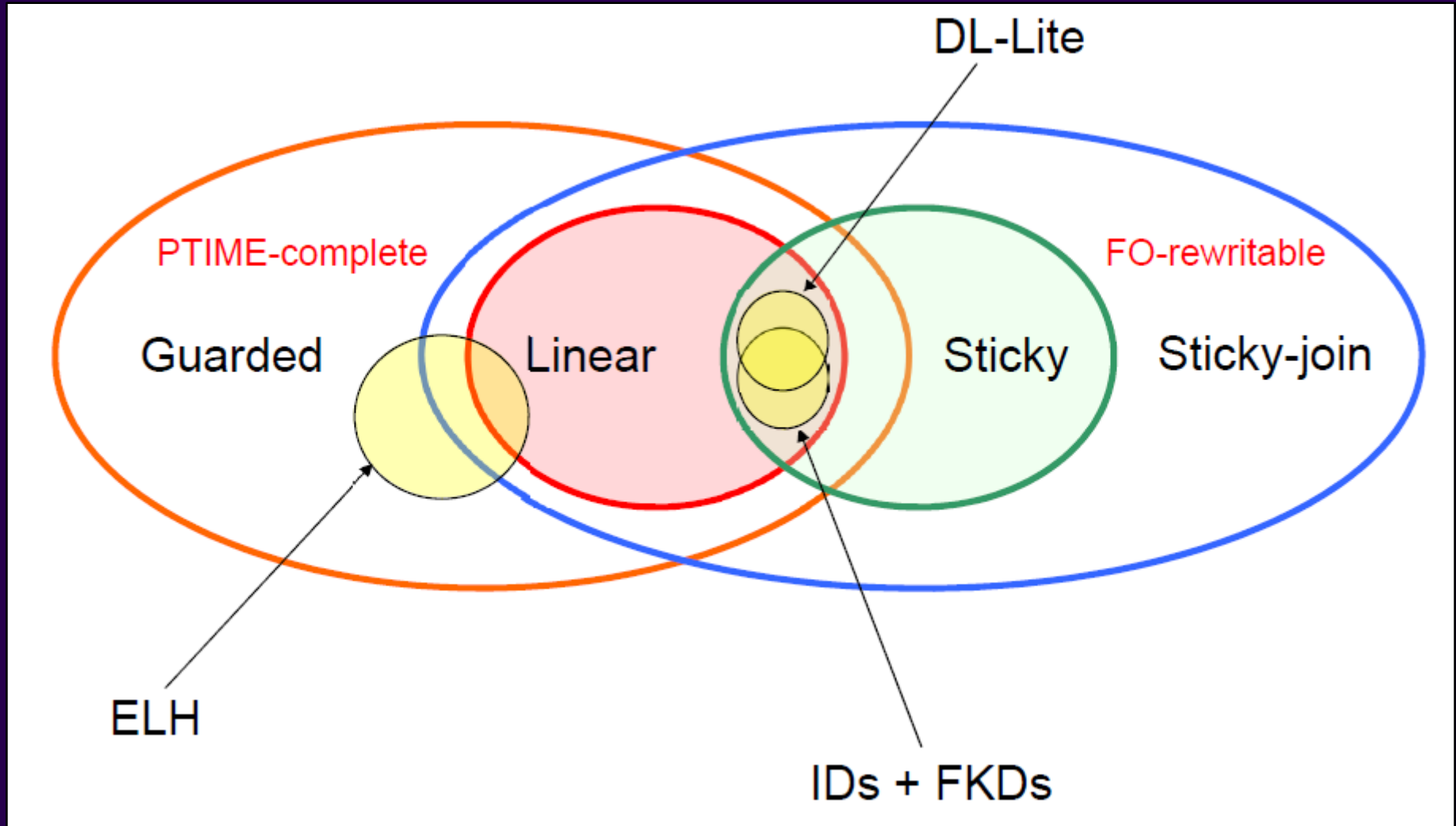
Otras propiedades

- EGDs: $\forall X \forall Y \forall Z \text{ reports}(X, Y), \text{ reports}(Y, Z) \rightarrow Y = Z$
 - *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*: $\forall X \text{ emp}(X), \text{ customer}(X) \rightarrow \perp$
 - Se puede *verificar* si $D \cup \Sigma$ satisface el conjunto de NCs sin agregar complejidad.

Resumen Expresividad de TGDs



Resumen Expresividad TGDs



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.
- 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-Marcuska.*

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-Marcuska.*
- 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/>