Ingeniería de ontologías

Ontologías – OWL

FACULTAD DE INGENIERIA

Agenda

- **Revisión ontologías - lógica descriptiva**
- **Lenguaje OWL**
- **OWL – Ejercicio Protégé**

ONTOLOGIA = Una especificación formal y explícita de una conceptualización compartida [Studer98]

- **Conceptualización**: modelo abstracto de un objeto de estudio
- **Formal**: representable en un lenguaje interpretable por agentes automáticos
- **Explícita**: conceptos, relaciones y axiomas que restringen la interpretación de los conceptos son explícitamente declarados
- **Compartida**: se representa conocimiento consensuado

Bloques de construcción de una ontología

- **Instancias o individuos:** constantes

  Elementos, objetos atómicos del dominio

  María, Uruguay

- **Conceptos ó clases:** predicados unarios

  Conjuntos de elementos del dominio

  Persona, *Estudiante*, *País*

- **Relaciones ó roles:** predicados binarios

  Conjuntos de pares de elementos del dominio

  Ej.: $vive \subseteq Persona \times Pais$

- **Axiomas:** sentencias que son siempre verdaderas

  Afirmaciones sobre individuos, conceptos y roles

Estudiante es una subclase de *Persona*	$\textbf{\textit{Estudiante}} \sqsubseteq \textbf{\textit{Persona}}$
María es una instancia de *Persona*	$\textbf{\textit{Persona}}(\textbf{\textit{María}})$
María vive en *Uruguay*	$\textbf{\textit{vive}}(\textbf{\textit{María, Uruguay}})$

Lógica Descriptiva - Sintaxis

Partimos de un conjunto de **nombres de conceptos atómicos *A*, *B*,..** y **nombres de roles atómicos *R*, *S*,.., nombres de individuos *a, b*, …**

Cada lógica permite construir conceptos y axiomas con diferente expresividad:

$\mathcal{ALC}$: $\top$, $\bot$, $\sqcap$, $\sqcup$, $\exists$, $\forall$, $\neg$

$\mathcal{S}$: $\mathcal{ALC}$ + **roles transitivos** Trans(*R*)

A $\mathcal{S}$ se agregan constructores que se representan por diferentes letras:

$\mathcal{H}$: inclusión de roles $\mathcal{O}$: nominales {*a*} $\mathcal{I}$: roles inversos

$\mathcal{N}$: restricciones numéricas $\mathcal{Q}$: restrictions numéricas calificadas

$\mathcal{R}$: Dis(*R*, *S*) roles disjuntos Irr(*R*) roles irreflexivos

Aserciones de negación de roles: (*John*, *Mary*) : ¬*likes*,

Axiomas de inclusión de roles complejos: $R \circ S \sqsubseteq Q$, universal role *U*, $\exists R.Self$

Descripción de conceptos en lógica $\mathcal{ALCQ}$:

$C, D := \bot | \top | A | \neg C | C \sqcap D | C \sqcup D | \forall R.C | \exists R.C | >=nR.C$

Concepto vacío Todo el Universo

Lógica Descriptiva - Sintaxis

Conceptos en lógica $\mathcal{ALCQ}$:

C, D := ⊥|⊤|***A***|¬***C***|***C*** ⊓ ***D***|***C*** ⊔ ***D***|∀***R.C***|∃***R.C***|>=***n****R.C*|<=***n****R.C*

Conceptos atómicos: *Persona, Madre, Padre, Mujer* Rol atómico*: tieneHijo*

¬***Persona:*** ***c***onjunto de todos los elementos que **no** satisfacen el predicado *Persona* (no pertenecen a ese conjunto)

Persona ⊓ ***Mujer:*** conjunto de todos los elementos que satisfacen el predicado *Persona* y satisfacen el predicado *Mujer* (pertenecen a ambos conjuntos)

Padre ⊔ ***Madre:*** conjunto de todos los elementos que satisfacen el predicado *Padre* (pertenecen a ese conjunto) ó satisfacen el predicado *Madre*

∃***tieneHijo.Persona:*** conjunto de todos los elementos que **están vinculados** a **algún** elemento del concepto *Persona* a través del rol *tieneHijo*

∀***tieneHijo.Persona:*** conjunto de todos los elementos que, **si están vinculados** a algún elemento a través del rol *tieneHijo,* este elemento **debe pertenecer al concepto *Persona***. Si NO está vinculado a ningún elemento a través de *tieneHijo,* también satisface la condición.

>=***2tieneHijo.Persona:*** conjunto de todos los elementos que **están vinculados a por lo menos 2 elementos** del concepto *Persona* a través del rol *tieneHijo*

Lógica Descriptiva – Sintaxis - Rbox

$\mathcal{ALC}$: $\top$, $\bot$, $\sqcap$, $\sqcup$, $\exists$, $\forall$, $\neg$

$\mathcal{S}$: $\mathcal{ALC}$ + **roles transitivos** Trans(R)

A $\mathcal{S}$ se agregan constructores que se representan por diferentes letras:

$\mathcal{H}$: inclusión de roles $\mathcal{O}$: nominales $\{a\}$ $\mathcal{I}$: roles inversos

$\mathcal{N}$: restricciones numéricas $\mathcal{Q}$: restrictions numéricas calificadas

$\mathcal{R}$: Dis(R, S) roles disjuntos Irr(R) roles irreflexivos

Aserciones de negación de roles: (*John*, *Mary*) : ¬*likes*,

Axiomas de inclusión de roles complejos: $R \circ S \sqsubseteq Q$, universal role U, $\exists R.Self$

$\mathcal{H}$: $R \sqsubseteq S$ *tieneHijo* $\sqsubseteq$ *tieneDescendiente*

$\mathcal{O}$: {*a*} {*Uruguay*} $\mathcal{I}$: $S = R^-$ *Hijo* = *Padre*$^-$

$\mathcal{N}$: >=*nR.*$\top$ >=2*tieneHijo.*$\top$

$\mathcal{R}$: Dis(*R*, *S*) Dis(*esAmigoDe*, *esEnemigoDe*) Irr(*R*) Irr(*tieneHijo*)

$R \circ S \sqsubseteq Q$ *esHermano* $\circ$ *esPadre* $\sqsubseteq$ *esTio*

Roles transitivos: **Trans(*R*)** $\rightarrow$ $R \circ R \sqsubseteq R$

Lógica Descriptiva - Sintaxis

Base de conocimiento $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$

TBox $\mathcal{T}$: Conocimiento básico, descripción intensional sobre el dominio de un problema. $C \sqsubseteq D$

ABox $\mathcal{A}$: Conocimiento sobre individuos, descripción extensional sobre el dominio de un problema.

$C(a) \quad R(a, b) \quad a = b \quad a \neq b$

$\mathcal{T}$ = {*Mujer* ⊑ *Persona*, ***Persona*** ≡ ***Mujer*** ⊔ ***Hombre,***

Madre ≡ *Mujer* ⊓ ∃*tieneHijo.Persona*}

$C \equiv D \rightarrow C \sqsubseteq D$ **y** $D \sqsubseteq C$

$\mathcal{A}$ = {*Mujer*(*maria*), *tieneHijo*(*maria*, *diego*)

Lógica Descriptiva - Sintaxis

Base de conocimiento $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$

TBox $\mathcal{T}$: Conocimiento básico, descripción intensional sobre el dominio de un problema. $C \sqsubseteq D$

ABox $\mathcal{A}$: Conocimiento sobre individuos, descripción extensional, sobre el dominio de un problema. $C(a)$ $R(a, b)$ $a = b$ $a \neq b$

RBox $\mathcal{R}$: Conocimiento básico, descripción intensional sobre el conjunto de pares de elementos del dominio. $R \sqsubseteq S$ $\mathrm{Dis}(R, S)$ $R \circ S \sqsubseteq Q$

$Mujer \sqsubseteq Persona$

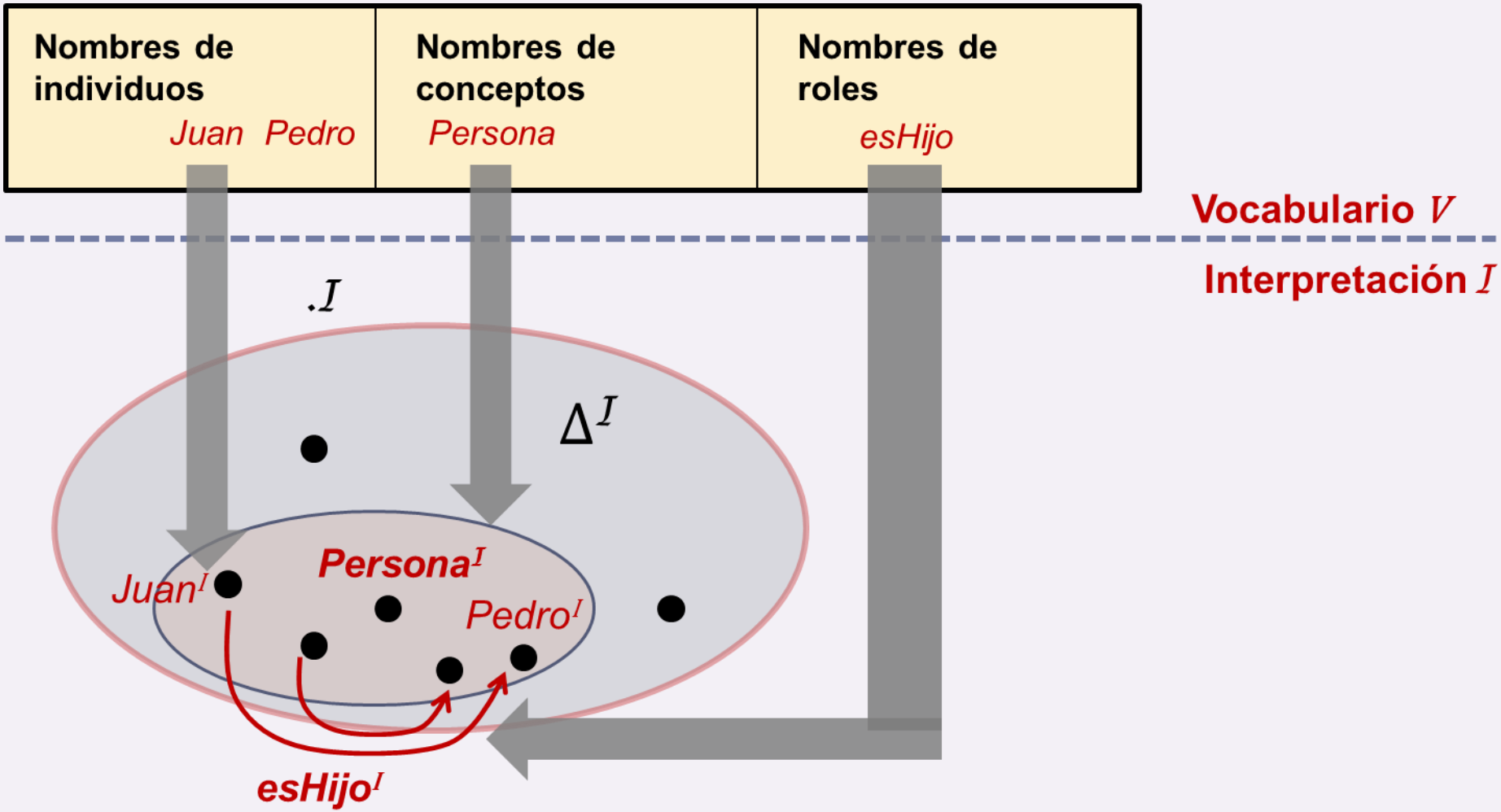
$tieneHijo(maria, diego)$

$mariajose = majo$

$esPadre \circ esPadre \sqsubseteq esAbuelo$

Lógica Descriptiva - Semántica

Interpretación



Web Ontology Language: OWL

Lenguajes de ontologías permiten escribir una **conceptualización formal y explícita de un modelo de dominio**.

- **Sintaxis bien definida**
- **Semántica bien definida**
- **Soporte eficiente de razonamiento**
- **Poder expresivo**

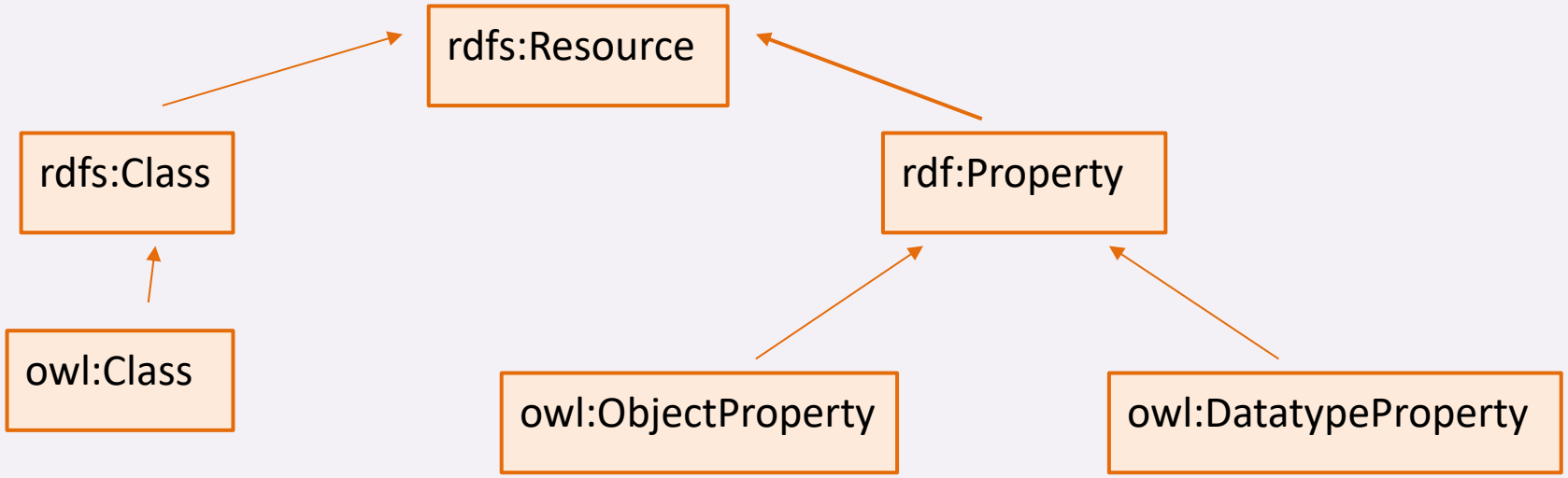
- Consistencia
- Subsumption de clases
- Clasificación de instancias

Web Ontology Language: OWL

OWL es un lenguaje de la Web Semántica para representar conocimiento complejo acerca de un dominio

Es un lenguaje basado en lógica, tal que es posible aplicar algoritmos de razonamiento para verificar consistencia y hacer explícito conocimiento implícito.

Constructores OWL



rdfs:subClass
rdfs:domain
rdfs:range
rdf:type

Instance

owl:differentFrom
owl:AllDifferent
owl:sameAs

Class

owl:disjointWith
owl:equivalentClass
owl:complementOf
owl:unionOf
owl:intersectionOf
owl:oneOf

Property

owl:Restriction
owl:minCardinality
owl:someValuesFrom
owl:allValuesFrom
owl:hasValue
owl:TransitiveProperty
owl:inverseOf
owl:SymmetricProperty
owl:equivalentProperty
owl:FunctionalProperty
owl:inverseFunctionalProperty

Constructores OWL - DL

OWL Constructor	DL
SubClassOf	$C \sqsubseteq D$
intersectionOf	$C \sqcap D$
unionOf	$C \sqcup D$
complementOf	$\neg C$
oneOf	$\{a_1, a_2, \ldots, a_n\}$
someValuesFrom	$\exists R.C$
allValuesFrom	$\forall R.C$
minCardinality	$\geq n.R$
maxCardinality	$\leq n.R$
cardinality	$= n.R$
hasValue	$\exists R.\{a\}$

OWL: Web Ontology Language

- 2 familias de OWL, difieren en la expresividad de la DL

inclusión de roles: $R \sqsubseteq S$ nominales: $\{a\}$

– **OWL 1** - $\mathcal{SHOIN}$ ----- number restrictions: >=3 *hasChild*

roles inversos: R^-

Conceptos, roles (atómicos), $\top$, $\bot$, $\sqcap$, $\sqcup$, $\exists$, $\forall$, $\neg$, roles transitivos

– **OWL 2** → $\mathcal{SROIQ}$ qualified number restrictions: >=3 *hasChild.Male*

Dis(*R*, *S*), (ir)reflexive roles, negated role assertions: ¬*likes* (*John*, *Mary*), complex role inclusion axioms: $R \circ S \sqsubseteq Q$, universal role *U*, constructs: $\exists R.Self$

OWL profiles

- OWL Lite: Fácil de usar y de implementar herramientas
- OWL DL: Decidible (todas las conclusiones son realizables en un tiempo finito)
- OWL Full: Máxima expresividad sin garantía de computabilidad.

Fragmentos OWL DL

- OWL 2 EL: Para ontologías de gran tamaño, complejidad polinomial

  $\top, \bot, \sqcap, \exists, \{a\}, \exists R.Self$, *keys*, $R \circ S \sqsubseteq Q$

- OWL 2 RL: Más expresivo que OWL 2 EL, complejidad polinomial

  $\top, \bot, \sqcap, \sqcup, \exists, \{a\} \sqsubseteq \top, \bot, \sqcap, \forall, \neg, <=1R.C$

  $R \circ S \sqsubseteq Q$

- OWL 2 QL: Orientado a consultas de grandes volúmenes de instancias, complejidad polinomial

  $\top, \bot, \exists R.\top \sqsubseteq \top, \bot, \sqcap, \neg, \exists$

  $R \sqsubseteq S, R^-$

Protégé: OWL Ontology Development Environment

- Download: http://protege.stanford.edu/

Sintaxis OWL-DL-Manchester

OWL Constructor	DL	Manchester Syntax (Protégé)
SubClassOf	$C \sqsubseteq D$	SubClassOf
intersectionOf	$C \sqcap D$	*C* and *D*
unionOf	$C \sqcup D$	*C* or *D*
complementOf	$\neg C$	not *C*
oneOf	$\{a_1, a_2, \ldots, a_n\}$	$\{a_1, a_2, \ldots, a_n\}$
someValuesFrom	$\exists R.C$	*R* some *C*
allValuesFrom	$\forall R.C$	*R* only *C*
minCardinality	$\geq n.R$	*R* min *n*
maxCardinality	$\leq n.R$	*R* max *n*
cardinality	$= n.R$	*R* exactly *n*
hasValue	$\exists R.\{a\}$	*R* value *a*

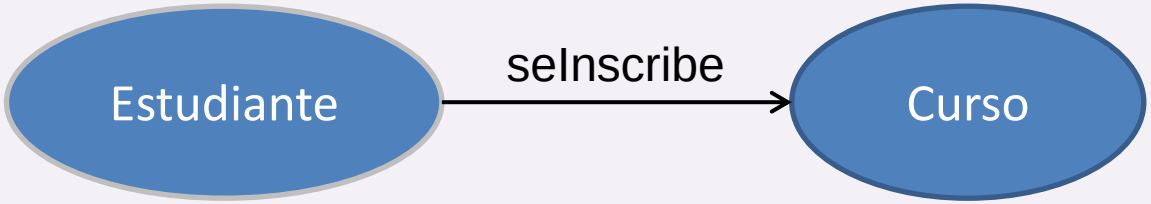
Dominio y rango

El dominio de la propiedad seInscribe es Estudiante

$\exists seInscribe.\top \sqsubseteq Estudiante$

El rango de la propiedad seInscribe es Curso

$\top \sqsubseteq \forall seInscribe.Curso$



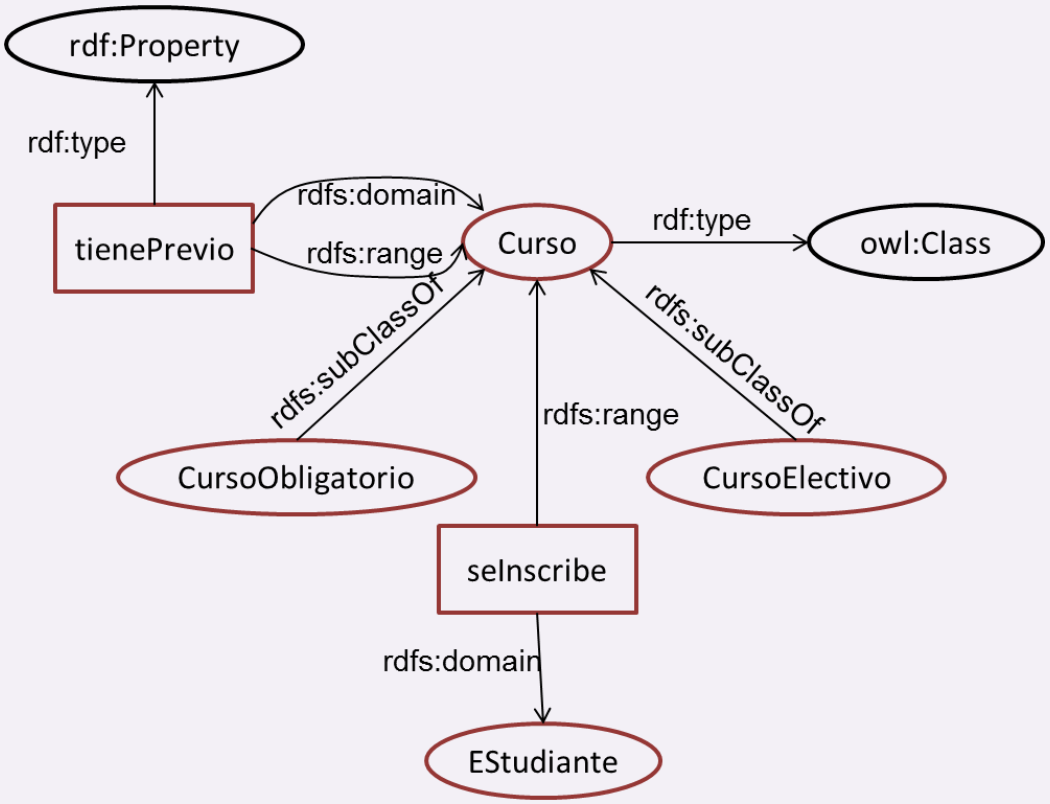
Ejercicio

Una de las actividades de la bedelía de la facultad es la inscripción de estudiantes a los cursos que se dictan el presente año.

Cada curso puede tener uno o más cursos previos.

Los cursos se clasifican en: obligatorios y electivos.

Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos.



Ejercicio

Una de las actividades de la bedelía de la facultad es la inscripción de estudiantes a los cursos que se dictan el presente año.

Cada curso puede tener uno o más cursos previos.

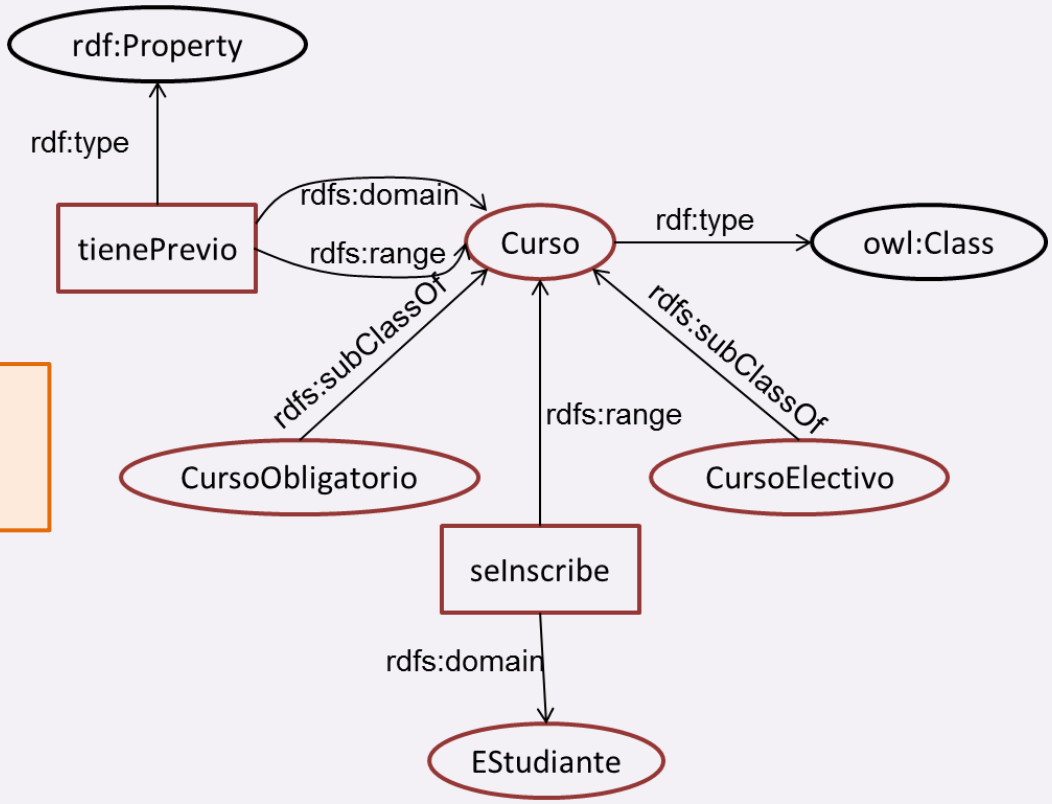
Los cursos se clasifican en: obligatorios y electivos.

Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos.

$$CursoObligatorio \sqcap CursoElectivo \sqsubseteq \bot$$

$$Curso \equiv CursoObligatorio \sqcup CursoElectivo$$

$$Estudiante \sqsubseteq <=2 seInscribe.CursoElectivo$$



Ejercicio

Una de las actividades de la bedelía de la facultad es la inscripción de estudiantes a los cursos que se dictan el presente año.

Cada curso puede tener uno o más cursos previos.

Los cursos se clasifican en: obligatorios y electivos.

Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos.

$$CursoObligatorio \sqcap CursoElectivo \sqsubseteq \bot$$

$$Curso \equiv CursoObligatorio \sqcup CursoElectivo$$

$$Estudiante \sqsubseteq <=2 seInscribe.CursoElectivo$$

A los efectos de producir datos estadísticos en bedelía se necesita obtener la siguiente información:

- **Una lista de todos los cursos que tengan tres o más cursos previos.**
- **Una lista con todos los estudiantes inscriptos a cursos obligatorios y a cursos electivos.**

Ejercicio

Una de las actividades de la bedelía de la facultad es la inscripción de estudiantes a los cursos que se dictan el presente año.

Cada curso puede tener uno o más cursos previos.

Los cursos se clasifican en: obligatorios y electivos.

Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos.

$$CursoObligatorio \sqcap CursoElectivo \sqsubseteq \bot$$

$$Curso \equiv CursoObligatorio \sqcup CursoElectivo$$

$$Estudiante \sqsubseteq <=2 seInscribe.CursoElectivo$$

A los efectos de producir datos estadísticos en bedelía se necesita obtener la siguiente información:

- **Una lista de todos los cursos que tengan tres o más cursos previos.**
- **Un lista con todos los estudiantes inscriptos a cursos obligatorios y a cursos electivos.**

$$CursoMinTresPr \equiv Curso \sqcap >=3 tienePrevio.Curso$$

$$EstudianteOblElect \equiv Estudiante \sqcap \exists seInscribe.CursoObligatorio \sqcap \exists seInscribe.CursoElectivo$$

Ejercicio

Ejecutar el razonador.

Ingresar individuos para: estudiantes, cursos obligatorios y cursos electivos.

Para cualquiera de los individuos de estudiantes, ingresar una inscripción para un curso obligatorio y otra para un curso electivo.

Para cualquiera de los individuos de estudiantes, ingresar inscripciones para más de dos cursos electivos.

Para cualquiera de los individuos de cursos (obligatorios o electivos), ingresar varias instancias que lo vinculen a 3 o más cursos previos.

Ejecutar el razonador y observar las inferencias generadas.

Se obtuvieron las inferencias esperadas?

Ejercicio (mundo abierto)

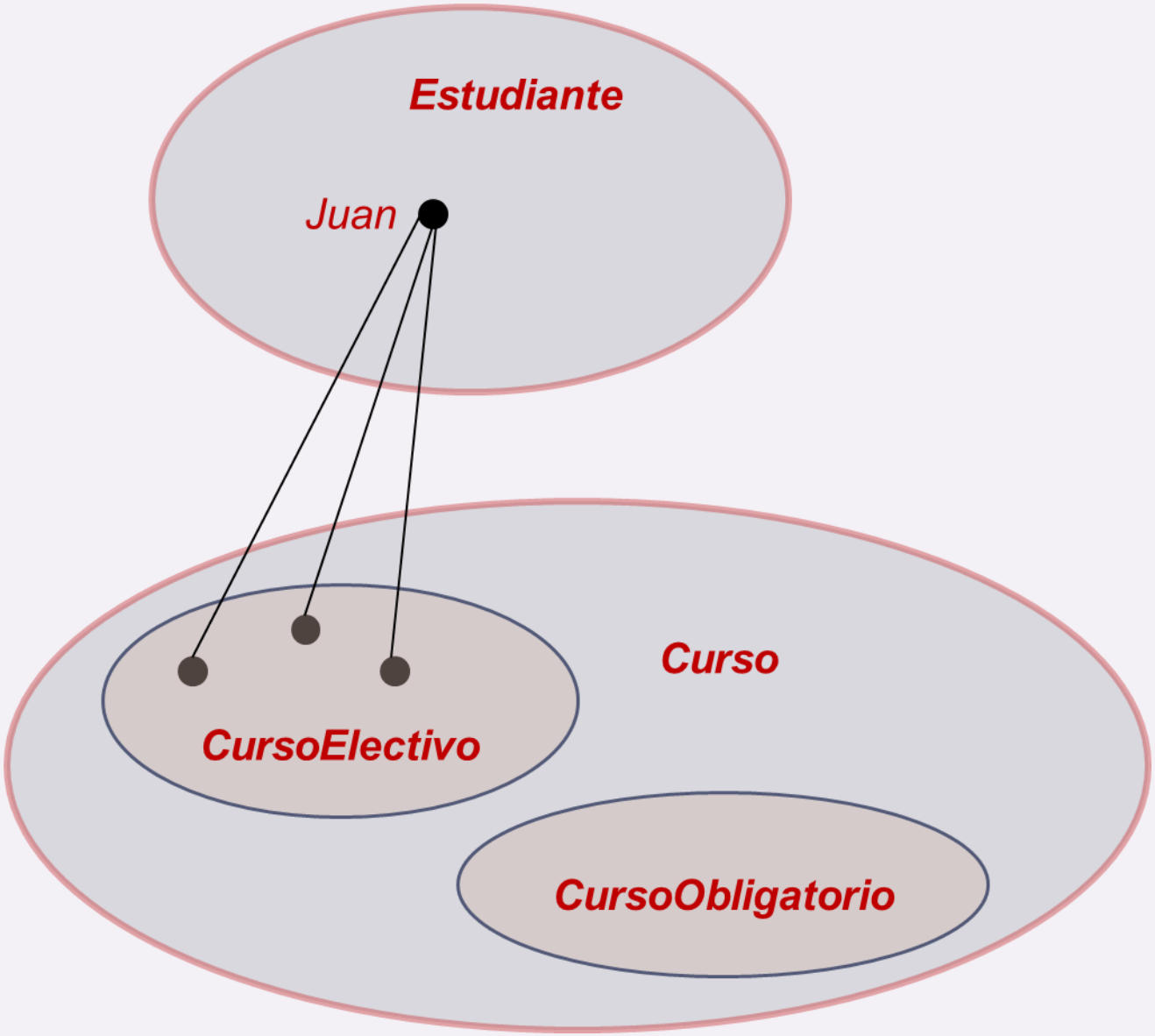
No se detectó inconsistencia:

“Cada estudiante puede inscribirse a cualquier cantidad de cursos obligatorios pero a no más de dos cursos electivos”

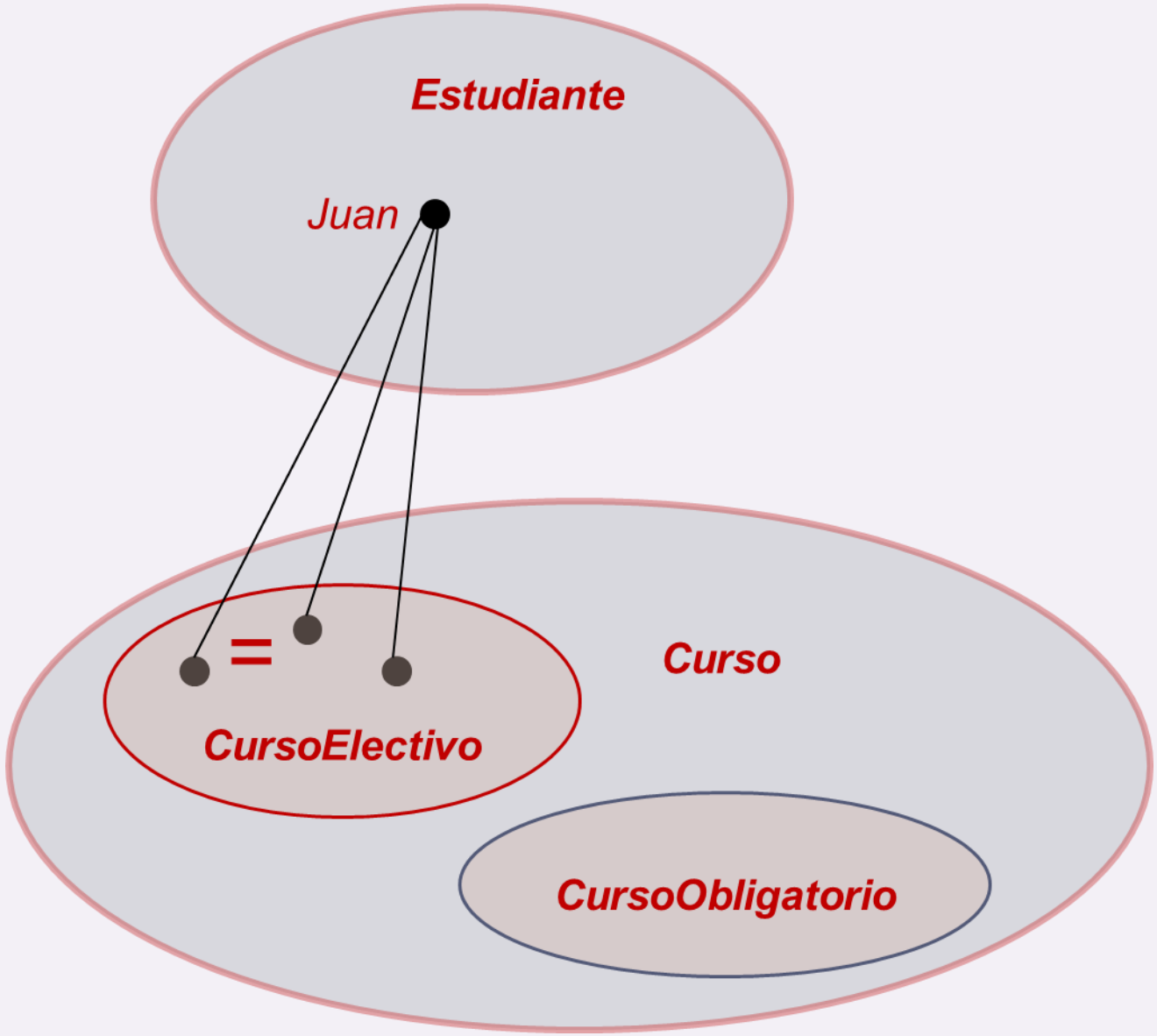
No se clasificaron los individuos de cursos que satisfacen:

“Una lista de todos los cursos que tengan tres o más cursos previos”

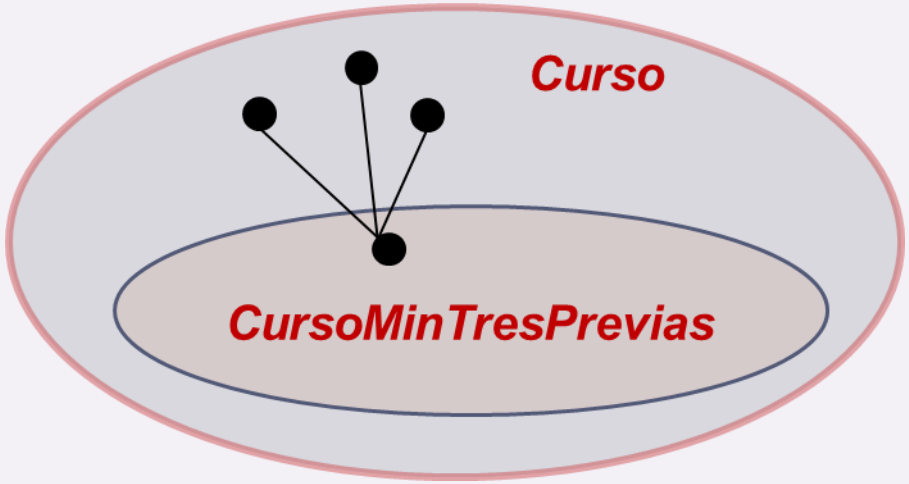
Ejercicio (mundo abierto)



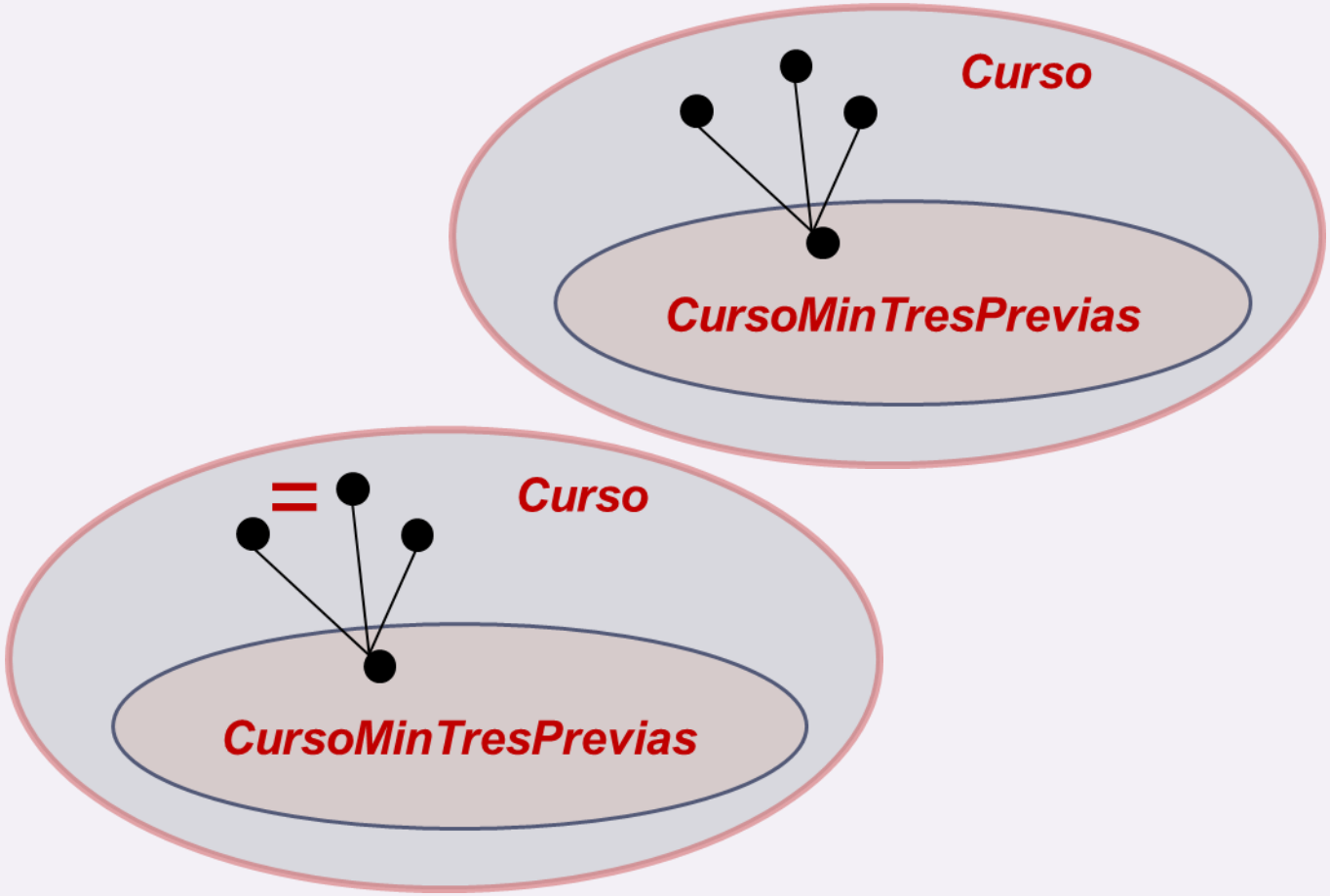
Ejercicio (mundo abierto)



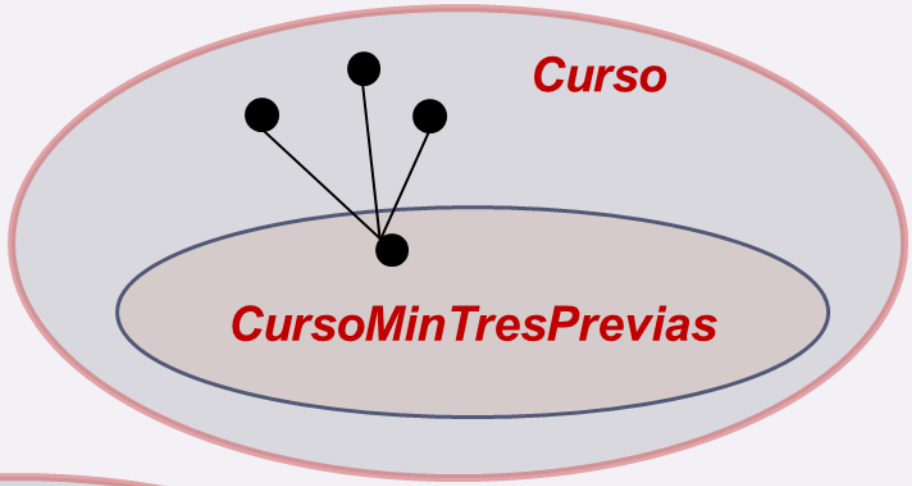
Ejercicio (mundo abierto)

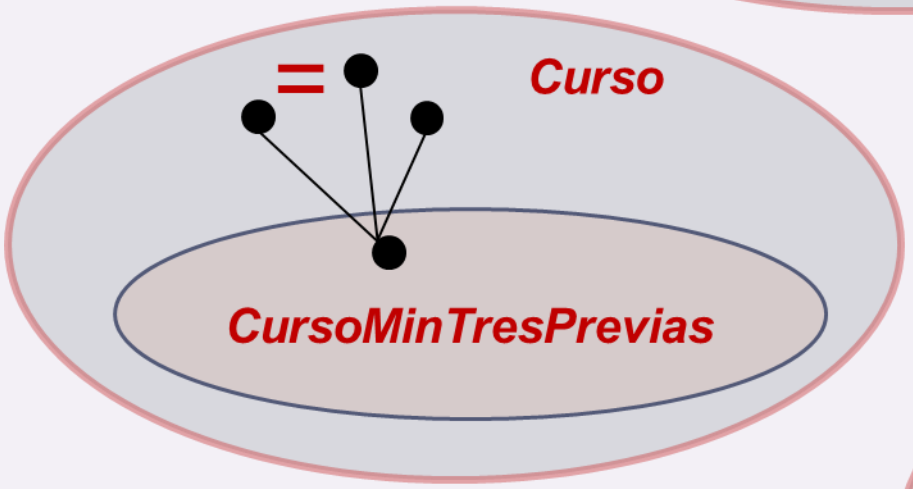


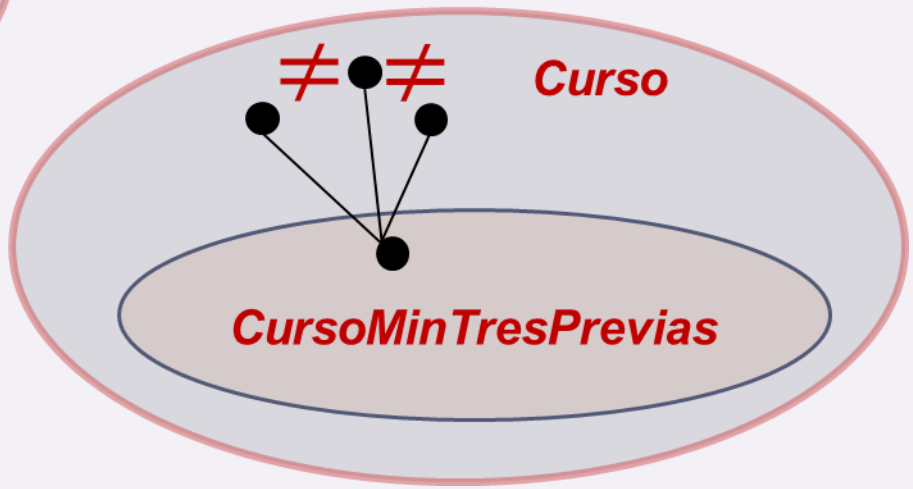
Ejercicio (mundo abierto)



Ejercicio (mundo abierto)







Ejercicio

Agregar al modelo que los estudiantes “aprueban” exámenes y que los exámenes “corresponden a” cursos.

Un estudiante “salva” un curso cuando aprueba el examen correspondiente a dicho curso.

Además, representar que cada estudiante tiene un único número de estudiante.

Para cualquiera de los individuos de estudiantes ingresar una instancia “aprueba” para un individuo de examenes y para éste ingresar una instancia “corresponde a” para un individuo de cursos.

Para uno de los individuos de estudiantes ingresar una instancia para asignarle un número de estudiante.

Ejecutar el razonador y observar las inferencias generadas.

Asociar al otro individuo de estudiantes el mismo número de estudiante.

Ejecutar el razonador y observar las inferencias generadas.

Protégé-OWL API

Open-source Java library para Web Ontology Language y RDF(S).

Permite cargar y guardar archivos OWL, consultar y manipular OWL data models, y ejecutar razonamiento.

http://protege.stanford.edu/plugins/owl/api/

- Protégé-OWL API: guía para el desarrollador

  http://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide

- Ejemplos de construcción de aplicaciones semánticas

  http://protegewiki.stanford.edu/wiki/BuildingSemanticWebApplications

Protégé-OWL API

- **Crear un modelo OWL a partir de un archivo .owl.**

  String uri = "file:/c:/Facultad/Postgrado%20Pedeciba/Proyectos/Proyecto%20SALUS/Ontologies/WebSiteSpecialization.owl";

  OWLModel owlModel = **ProtegeOWL.*createJenaOWLModelFromURI***(uri);

- **Obtener una clase del modelo:**

  OWLNamedClass WebContentClass = **owlModel.getOWLNamedClass**("http://www.owl-ontologies.com/WebSite#WebContent");

  OWLNamedClass AuthorClass = **owlModel.getOWLNamedClass(**"http://www.owl-ontologies.com/WebSiteSpecialization#Author");

- **Obtener una object property:**

  OWLObjectProperty hasAuthorProperty =

  owlModel.getOWLObjectProperty("http://www.owl-ontologies.com/WebSiteSpecialization#hasAuthor");

- **Crear una instancia:**

  OWLIndividual AlzheimerWebContent = **WebContentClass.createOWLIndividual**("AlzheimerWebContent");

  OWLIndividual JuanPerezAuthor **= AuthorClass.createOWLIndividual**("JuanPerezAuthor");

- **Asociar la instancia de una propiedad a una instancia dada:**

  AlzheimerWebContent.setPropertyValue(hasAuthorProperty, JuanPerezAuthor);

Protégé-OWL API

- **Crear una instancia del razonador:**

  ReasonerManager reasonerManager = **ReasonerManager.*getInstance*()**;

  ProtegeReasoner reasoner = **reasonerManager.createProtegeReasoner**(owlModel,ProtegePelletOWLAPIReasoner.**class**);

- **Controlar conceptos inconsistentes, clasificar conceptos e instancias:**

  reasoner.computeInconsistentConcepts();

  reasoner.classifyTaxonomy();

  reasoner.computeInferredIndividualTypes();

- **Obtener instancias clasificadas en un concepto:**

  Collection webContentIndividuals = **reasoner.getIndividualsBelongingToClass**(WebContentClass);

- **Obtener subclases inferidas:**

  Collection inferredSubclasses = **WebContentClass.getInferredSubclasses();**

Bibliografía

Handbook on Ontologies. Steffen Staab, Rudi Studer (Editors). 2004.

Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph: Foundations of Semantic Web Technologies, Chapman & Hall/CRC, 2009.

Capítulos 4 y 5.