# **Documento de Diseño**

## **Introducción**

Este documento describe la arquitectura, diseño e implementación de un sistema inteligente basado en tres componentes fundamentales: un sistema experto, un módulo de lógica difusa y una ontología semántica en RDF. El sistema tiene como objetivo recomendar recetas culinarias personalizadas teniendo en cuenta variables subjetivas (estado de ánimo, habilidad, tiempo), conocimiento explícito estructurado en una ontología y razonamiento experto codificado en reglas.

## **1. Sistema Experto**

### **1.1 Descripción General**

El sistema experto está diseñado para recomendar recetas de cocina personalizadas. Integra conocimientos de una ontología de recetas, preferencias explícitas del usuario, y un análisis del contexto del usuario (tiempo disponible, habilidad culinaria, estado de ánimo) mediante lógica difusa. Su objetivo es evaluar un conjunto de recetas base y puntuar aquellas que mejor se adapten a las necesidades y restricciones del usuario en un momento dado.

El sistema se implementa en Python utilizando las siguientes librerías clave:

* **experta:** Para el motor de inferencia basado en reglas (estilo CLIPS). Gestiona los hechos y dispara las reglas según coinciden los patrones.
* **rdflib:** Para cargar y consultar la ontología de recetas almacenada en formato Turtle (.ttl). Extrae la información estructural y semántica de las recetas.
* **scikit-fuzzy (skfuzzy) y numpy:** Para implementar el sistema de lógica difusa que estima el "nivel de esfuerzo" que el usuario está dispuesto o es capaz de invertir, basándose en su tiempo, habilidad y ánimo.

El flujo de trabajo general es:

1. **Inicialización:** Carga la ontología (.ttl) en un grafo RDF y configura el sistema de lógica difusa.
2. **Generación de Hechos Base:** Al iniciar el motor experta, el método @DefFacts (\_cargar\_hechos\_base\_ontologia) consulta el grafo RDF y declara hechos iniciales (Receta, TieneIngrediente, PropiedadReceta, etc.) para cada receta encontrada en la ontología.
3. **Consulta del Usuario:** El método recomendar recibe las preferencias del usuario (lista de diccionarios), su tiempo disponible, nivel de habilidad y estado de ánimo.
4. **Cálculo de Esfuerzo Difuso:** Utiliza el sistema difuso (\_crear\_sistema\_difuso) para calcular un NivelEsfuerzoUsuario estimado (valor numérico entre 0 y 10).
5. **Declaración de Hechos Dinámicos:** Se declaran los hechos PreferenciaUsuario basados en la entrada del usuario y el hecho NivelEsfuerzoUsuario calculado.
6. **Ejecución del Motor de Inferencia:** Se invoca self.run() para que el motor evalúe las reglas contra la base de hechos actual.
7. **Puntuación y Selección:** Las reglas activadas declaran hechos RecetaCandidata, asignando puntos (positivos o negativos) y motivos a las recetas que cumplen o incumplen ciertos criterios.
8. **Consolidación y Ordenación:** Después de ejecutar las reglas, el sistema consolida los puntajes y motivos para cada receta candidata, filtra las que tienen puntaje no negativo (>= 0) y las ordena por puntaje descendente (y luego por popularidad descendente como criterio secundario).
9. **Resultado:** Devuelve una lista de las recetas recomendadas, cada una con su URI, nombre, puntaje final, motivos de la recomendación y nivel de popularidad.

### **1.2 Clases de Hechos**

Se definen las siguientes clases de hechos (Fact) para representar la información manejada por el motor:

1. **Receta:** Información básica de una receta.
   1. uri: Identificador único (URI) de la receta en la ontología.
   2. nombre: Nombre legible de la receta.
2. **EsTipoReceta:** Indica la clasificación ontológica de una receta (ej. "Postre", "PlatoPrincipal").
   1. receta\_uri: URI de la receta.
   2. tipo: Nombre del tipo o clase de la receta.
3. **TieneIngrediente:** Asocia una receta con uno de sus ingredientes.
   1. receta\_uri: URI de la receta.
   2. ingrediente: Nombre del ingrediente.
4. **UsaInstrumento:** Asocia una receta con un instrumento de cocina necesario.
   1. receta\_uri: URI de la receta.
   2. instrumento: Nombre del instrumento.
5. **PerteneceATipoCocina:** Indica el tipo de cocina al que pertenece la receta (ej. "CocinaItaliana", "CocinaMexicana"). Incluye tipos directos y superclases inferidas de la ontología.
   1. receta\_uri: URI de la receta.
   2. tipo\_cocina: Nombre del tipo de cocina.
6. **PropiedadReceta:** Almacena propiedades específicas de una receta como pares clave-valor.
   1. receta\_uri: URI de la receta.
   2. propiedad: Nombre de la propiedad (ej. "tiempoPreparacion", "dificultad", "momentoDia", "nivelPopularidad").
   3. valor: Valor de la propiedad (puede ser numérico, string, etc.).
7. **PreferenciaUsuario:** Representa una preferencia o restricción del usuario.
   1. criterio: El aspecto sobre el que se define la preferencia (ej. "ingrediente\_deseado", "tipo\_cocina", "tiempo\_maximo", "dificultad\_maxima", "instrumento\_no\_deseado").
   2. valor: El valor deseado o no deseado para el criterio.
   3. operador: (Opcional, default "igual") Podría usarse para comparaciones más complejas (no usado extensivamente en las reglas actuales).
   4. peso: Factor de importancia (default 1) que modula el puntaje asignado por las reglas que coinciden con esta preferencia.
8. **RecetaCandidata:** Hecho intermedio generado por las reglas. Representa una evaluación parcial de una receta.
   1. receta\_uri: URI de la receta evaluada.
   2. nombre: Nombre de la receta.
   3. puntaje: Puntos asignados por una regla específica (se acumulan al final).
   4. motivos: Lista de strings explicando por qué se asignó ese puntaje (ej. "Contiene tomate", "Tiempo prep. 30 min <= 45 (OK)").
9. **RecetaSugerida:** Hecho final que representa una receta recomendada después de la consolidación y filtrado.
   1. receta\_uri: URI de la receta.
   2. nombre: Nombre de la receta.
   3. puntaje: Puntaje final consolidado.
   4. motivos: Lista consolidada y única de motivos.
   5. popularidad: Nivel de popularidad de la receta (extraído de PropiedadReceta).
10. **Info:** Para mensajes internos del sistema (ej., advertencias, notificaciones).
    1. mensaje: El texto del mensaje.
11. **EstadoConsulta:** Hecho utilizado como marcador para control de flujo, principalmente para evitar que la misma regla se dispare múltiples veces sobre la misma combinación de hechos en una sola ejecución (run).
    1. procesada: String único que identifica qué combinación ha sido procesada (ej. f"ing\_deseado\_{uri}\_{ing\_pref}").
12. **NivelEsfuerzoUsuario:** Almacena el resultado numérico del sistema de lógica difusa.
    1. valor: El nivel de esfuerzo estimado (0-10).

### **1.3 Reglas del Sistema Experto**

El sistema define 16 reglas (@Rule) que evalúan las recetas en base a los hechos disponibles (recetas, propiedades, preferencias, esfuerzo). Estas reglas asignan puntajes positivos (bonificaciones) o negativos (penalizaciones) a las RecetaCandidata.

#### **Lógica de las Reglas del Sistema Experto:**

#### **Ingrediente Deseado:** Suma puntos (ponderados) si la receta contiene un ingrediente preferido.

#### **Ingrediente No Deseado:** Resta muchos puntos (ponderados) si la receta contiene un ingrediente a evitar.

#### **Tipo de Cocina:** Suma puntos (ponderados) si la receta es del tipo de cocina preferido.

#### **Tipo de Receta:** Suma puntos (ponderados) si la receta es del tipo específico (clase) deseado.

#### **Momento del Día:** Suma puntos (ponderados) si la receta encaja en el momento del día preferido.

#### **Cumple Tiempo Máximo:** Suma puntos (ponderados) si el tiempo de preparación es menor o igual al máximo del usuario.

#### **Excede Tiempo Máximo:** Resta muchos puntos (ponderados) si el tiempo de preparación supera el máximo del usuario.

#### **Cumple Dificultad Máxima Explícita:** Suma pocos puntos (ponderados) si la dificultad no supera el máximo explícito del usuario.

#### **Excede Dificultad Máxima Explícita:** Resta puntos (ponderados) si la dificultad supera el máximo explícito del usuario.

#### **Dificultad Compatible (Fuzzy):** Suma puntos si la dificultad es menor o igual al esfuerzo estimado por lógica difusa.

#### **Dificultad Excede (Fuzzy):** Resta puntos (proporcional a la diferencia) si la dificultad supera el esfuerzo estimado por lógica difusa.

#### **Instrumento No Deseado:** Resta muchísimos puntos (ponderados) si la receta usa un instrumento que el usuario quiere evitar.

#### **Popularidad:** Suma unos pocos puntos extra si la receta tiene algún nivel de popularidad registrado.

#### **No Hay Recetas:** Si ninguna receta candidata tiene puntaje final >= 0, genera un mensaje informativo.

#### **Mostrar Info:** Imprime cualquier mensaje informativo generado por el sistema.

#### **Técnicas de control de ejecución usadas:**

#### **Salience (Prioridad de Regla):** Se asignan diferentes niveles de prioridad (valores entre -100 y 35) para controlar el orden en que se evalúan y disparan las reglas. Las reglas con mayor salience se consideran primero. Esto es crucial para aplicar penalizaciones fuertes (como ingredientes o instrumentos no deseados) antes de sumar puntos por aspectos positivos.

#### **Activación Bloqueante (mediante NOT y Hechos de Estado):** Cada regla de puntuación incluye una condición NOT(EstadoConsulta(procesada=...)) donde el string procesado es único para la combinación específica de receta y criterio/preferencia que la regla está evaluando. Cuando la regla se dispara, declara el hecho EstadoConsulta correspondiente. Esto evita que la misma regla se vuelva a disparar para la misma combinación dentro de la misma ejecución de self.run(), previniendo bucles y puntuaciones duplicadas por la misma razón.

#### **Condiciones TEST:** Se usan funciones lambda dentro de TEST para realizar comparaciones numéricas (ej. tiempo <= tiempo\_maximo, dificultad > esfuerzo\_estimado) o validaciones de tipo (isinstance(valor, int)) directamente en la parte condicional de la regla.

#### **Consolidación Post-Ejecución:** El método recomendar realiza una consolidación explícita de los hechos RecetaCandidata después de que el motor (self.run()) ha terminado. Suma los puntajes parciales y agrupa los motivos para cada URI de receta única. Esto maneja la situación donde múltiples reglas (positivas y negativas) pueden afectar a la misma receta.

#### **Prevención de conflictos:**

* El uso de EstadoConsulta previene que una regla específica contribuya múltiples veces al puntaje de una receta por la misma razón exacta durante una ejecución.
* La estrategia de consolidación suma algebraicamente los puntajes positivos y negativos generados por diferentes reglas para la misma receta, resolviendo el "conflicto" de múltiples evaluaciones sobre un mismo ítem.
* El filtrado final por puntaje (>= 0) asegura que solo las recetas que globalmente cumplen los criterios (o al menos no los incumplen gravemente) sean sugeridas.

### **1.4 Base de Hechos**

La base de hechos (Fact-base) del sistema es dinámica y se compone de:

1. **Hechos Estáticos (Cargados al inicio desde la Ontología):**
   1. Generados por el método @DefFacts \_cargar\_hechos\_base\_ontologia.
   2. Incluyen:
      1. Receta (una por cada receta en la ontología).
      2. EsTipoReceta, TieneIngrediente, UsaInstrumento, PerteneceATipoCocina, PropiedadReceta (múltiples hechos de estos tipos, describiendo los detalles de cada Receta).
   3. El número de estos hechos depende directamente del contenido del archivo ontologia\_recetas2.ttl. Podrían ser cientos o miles de hechos si la ontología es grande (ej. 20 recetas con 10 ingredientes y 5 propiedades cada una ya sumarían ~320 hechos iniciales).
2. **Hechos Dinámicos (Declarados durante la consulta recomendar):**
   1. **PreferenciaUsuario:** Uno o más hechos, según las preferencias proporcionadas por el usuario en la llamada a recomendar.
   2. **NivelEsfuerzoUsuario:** Un solo hecho con el valor calculado por el sistema difuso.
3. **Hechos Inferidos (Declarados por las reglas durante run):**
   1. **RecetaCandidata:** Múltiples hechos, uno por cada vez que una regla de puntuación se dispara para una receta.
   2. **EstadoConsulta:** Múltiples hechos, usados como marcadores internos por las reglas.
   3. **Info:** Ocasionalmente, si se detectan condiciones especiales (como no encontrar recetas).
   4. **RecetaSugerida:** Declarados al final del proceso recomendar para representar el resultado final (opcional para la lógica principal, pero útil para trazabilidad).

La base de hechos combinada es la que el motor de experta utiliza para encontrar coincidencias con los patrones de las reglas y ejecutar el proceso de inferencia que lleva a las recomendaciones finales.

## **2. Lógica Difusa**

### **2.1. Descripción**

El sistema de lógica difusa fue construido utilizando la librería scikit-fuzzy, una herramienta que permite modelar y operar sistemas de inferencia difusa basados en lógica borrosa. Su objetivo principal es calcular una variable de salida llamada esfuerzo, que representa el nivel físico y mental que una receta de cocina puede exigirle al usuario, considerando condiciones subjetivas como el tiempo disponible, el estado emocional y la experiencia culinaria.

Este módulo permite suavizar las transiciones entre decisiones y ajustar la recomendación de recetas a las capacidades y estado actual del usuario, en lugar de utilizar reglas rígidas. La lógica difusa es especialmente útil en contextos donde no existen límites exactos, sino gradaciones de percepción.

### **2.2. Variables de Entrada**

El sistema considera tres variables de entrada, cada una modelada mediante un conjunto de términos lingüísticos definidos con funciones de membresía apropiadas:

#### **• tiempo (rango: 0–120 minutos)**

Representa la cantidad de tiempo que el usuario dispone para cocinar:

* **Corto**: trapmf([0, 0, 15, 30])
* **Medio**: trimf([20, 45, 70])
* **Largo**: trapmf([60, 90, 120, 120])
* **Ligeramente corto** *(modificador lingüístico aplicado)*: raíz cuadrada de trapmf([0, 0, 15, 30])

#### **• habilidad (rango: 0–10)**

Indica el nivel de experiencia culinaria del usuario:

* **Principiante**: gaussmf(2, σ=1.5)
* **Intermedio**: trimf([3, 5, 7])
* **Experto**: trapmf([6, 8, 10, 10])
* **Muy experto** *(modificador lingüístico aplicado)*: cuadrado de trapmf([6, 8, 10, 10])

#### **• estado\_animo (rango: 0–10)**

Evalúa el estado emocional del usuario en el momento:

* **Estresado**: trapmf([0, 0, 3, 5])
* **Neutral**: trimf([3, 5, 7])
* **Motivado**: trapmf([5, 8, 10, 10])
* **Muy motivado** *(modificador lingüístico aplicado)*: cuadrado de trapmf([5, 8, 10, 10])

Estos modificadores (muy, ligeramente) permiten intensificar o atenuar la pertenencia a ciertos estados, enriqueciendo el comportamiento del sistema.

### **2.3. Variable de Salida**

#### **• esfuerzo (rango: 0–10)**

Es la salida del sistema difuso, que representa el nivel total de esfuerzo requerido por una receta.

* **Bajo**: triangular [0, 0, 4]
* **Medio**: triangular [2, 5, 8]
* **Alto**: triangular [6, 10, 10]

El valor obtenido de esta variable será utilizado como **hecho de entrada** para el sistema experto, condicionando las recomendaciones.

### **2.4. Reglas Difusas**

El sistema de inferencia difusa utiliza un conjunto de **nueve reglas** tipo **IF–THEN**, diseñadas para capturar de manera flexible las combinaciones posibles entre las variables de entrada (tiempo, habilidad, estado\_ánimo) y producir como salida un valor representativo del **nivel de esfuerzo culinario** requerido (esfuerzo).

Estas reglas fueron formuladas considerando tanto condiciones normales como modificadas mediante operadores lingüísticos como “muy” o “ligeramente”, así como operadores lógicos difusos (AND, OR, NOT).

A continuación, se detallan las reglas implementadas:

1. **Si** el tiempo es **corto** **O** la habilidad es **principiante** **O** el estado\_ánimo es **estresado**,  
    **entonces** el esfuerzo es **bajo**.  
    *Regla general que capta cualquier limitación física, emocional o de disponibilidad.*
2. **Si** el tiempo es **largo** **Y** la habilidad es **experto** **Y** el estado\_ánimo es **motivado**,  
    **entonces** el esfuerzo es **alto**.  
    *Representa un escenario ideal donde el usuario está capacitado y dispuesto.*
3. **Si** la habilidad es **intermedio** **Y** el tiempo es **medio**,  
    **entonces** el esfuerzo es **medio**.  
    *Un caso equilibrado y común en usuarios promedio.*
4. **Si** el estado\_ánimo es **motivado** **Y** el tiempo es **largo**,  
    **entonces** el esfuerzo es **medio**.  
    *El estado emocional impulsa el esfuerzo, aunque no sea experto.*
5. **Si** la habilidad es **principiante** **Y** el estado\_ánimo es **estresado**,  
    **entonces** el esfuerzo es **bajo**.  
    *Situación vulnerable que justifica sugerencias fáciles.*
6. **Si** el estado\_ánimo es **muy motivado** **Y** la habilidad es **experto**,  
    **entonces** el esfuerzo es **alto**.  
    *Caso de entusiasmo elevado, usando modificador lingüístico "muy".*
7. **Si** el tiempo es **ligeramente corto** **Y** la habilidad es **intermedio**,  
    **entonces** el esfuerzo es **medio**.  
    *Modelo suavizado mediante el modificador "ligeramente".*
8. **Si** el tiempo **no es largo** **Y** el estado\_ánimo es **estresado**,  
    **entonces** el esfuerzo es **bajo**.  
    *Regla con negación (NOT), ante falta de condiciones ideales.*
9. **Si** el tiempo es **medio** **Y** la habilidad es **muy experto**,  
    **entonces** el esfuerzo es **alto**.  
    *El sistema reconoce la alta capacidad del usuario, aunque el tiempo no sea máximo.*

### **2.5. Modificadores Lingüísticos**

Para enriquecer las condiciones, se implementaron **modificadores lingüísticos** que transforman las funciones de membresía originales:

* **Muy**: Eleva al cuadrado los valores de pertenencia (μ(x)^2), haciendo más estricta la condición.  
   Ejemplo: “muy motivado” significa que el usuario debe estar en un estado emocional intensamente positivo.
* **Ligeramente**: Aplica la raíz cuadrada de los valores (√μ(x)), suavizando la condición.  
   Ejemplo: “ligeramente corto de tiempo” permite considerar tiempos algo mayores que un “corto” tradicional.

Estos modificadores permiten una mayor expresividad en las reglas, mejorando la adaptabilidad del sistema.

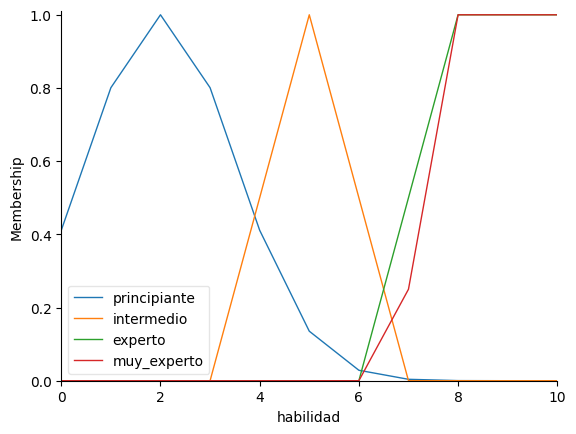
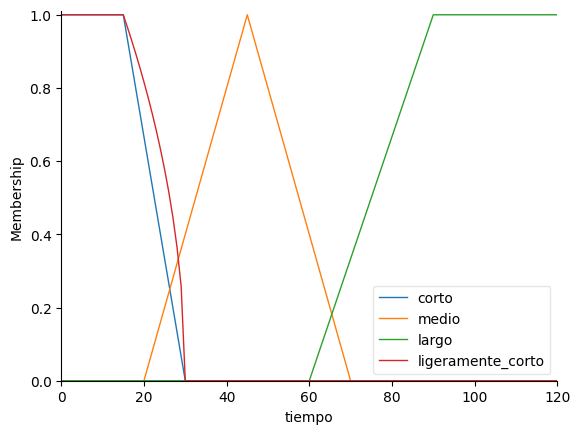
### **2.6. Método de Defuzzificación**

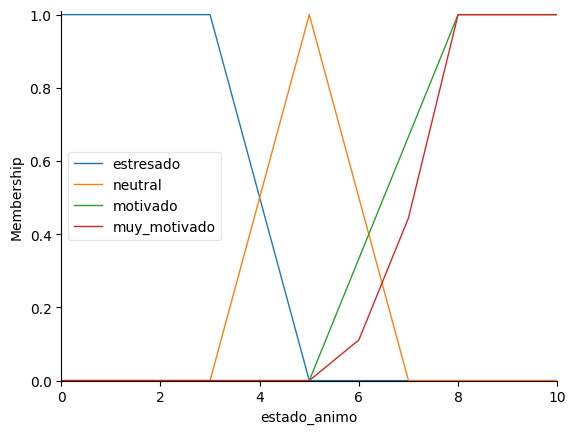
Para transformar los resultados del sistema de inferencia difuso en una salida numérica concreta, se utilizó el **método del centroide**, también conocido como **centro de gravedad**. Este método calcula el valor promedio ponderado del área bajo la curva de pertenencia resultante. Su elección se justifica por las siguientes razones:

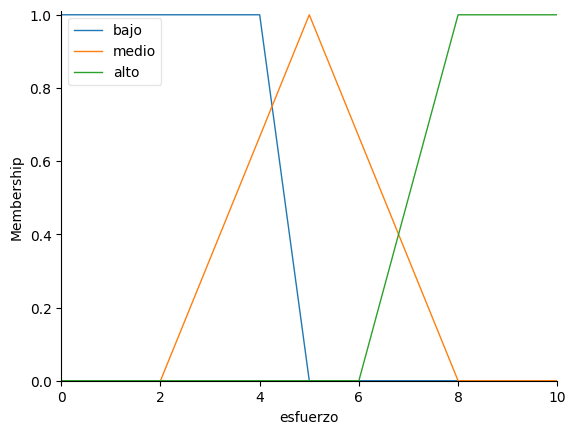
1. **Representación integral del razonamiento difuso** El sistema puede activar múltiples reglas simultáneamente con distintos grados de pertenencia. El centroide permite combinar todos estos valores de forma equilibrada, proporcionando una salida que refleja el razonamiento global del sistema y no una única condición dominante.
2. **Generación de resultados continuos y realistas** En el contexto culinario, no es deseable una respuesta tajante (por ejemplo, clasificar una receta como exclusivamente “fácil” o “difícil”). El centroide permite obtener resultados intermedios (como 4.3 o 6.8), lo que permite adaptar la sugerencia de receta con mayor precisión a las condiciones del usuario.
3. **Correspondencia con la experiencia humana** El razonamiento humano rara vez es binario. Un usuario que se siente “ligeramente estresado” y tiene “algo de tiempo” no espera una clasificación extrema. El centroide modela correctamente estos estados intermedios, generando una experiencia más empática y natural.
4. **Equilibrio entre precisión e interpretabilidad** A diferencia de métodos como el de **máximo-mínimo**, que seleccionan únicamente el pico más alto, el centroide considera todas las contribuciones de las reglas activadas. Esto mejora la precisión sin sacrificar la comprensibilidad del resultado, algo esencial en sistemas inteligentes destinados a la interacción humana.

En resumen, el uso del centroide permite tomar decisiones más humanas, ajustadas y razonadas, lo que es fundamental para sistemas que buscan ofrecer recomendaciones personalizadas y flexibles como el que se propone en esta práctica.

### **2.7. Gráficas de las funciones de pertenencia**







## **3. Ontología**

En este apartado se presenta la documentación entregable del **módulo de Ontologías**, siguiendo los requerimientos de la práctica y cubriendo todos los aspectos solicitados:

### **3.1 Descripción General**

El módulo de ontología está implementado en Python utilizando **RDFLib** para la creación y gestión de un grafo RDF y **OWL‑RL** (a través de la librería owlrl) para aplicar razonamiento semántico RDFS. Esta ontología modela el dominio culinario (recetas, ingredientes, utensilios, tipos de cocina, momentos del día, niveles de dificultad, popularidad, etc.) de forma estructurada y permite generar nuevo conocimiento mediante reglas de inferencia.

### **3.2 Clases Principales e Infraestructura**

Se definieron **12 clases** (todas implementadas como rdfs:Class):

1. Receta
2. Postre (subclase de Receta)
3. Desayuno (subclase de Receta)
4. Almuerzo (subclase de Receta)
5. Cena (subclase de Receta)
6. Ingrediente
7. Instrumentos
8. TipoCocina
9. CocinaLatina (subclase de TipoCocina)
10. CocinaEuropea (subclase de TipoCocina)
11. CocinaOriental (subclase de TipoCocina)
12. CocinaFusion (subclase directa de TipoCocina)

Se crearon **7 relaciones jerárquicas** mediante rdfs:subClassOf:

* Postre ⊑ Receta
* Desayuno ⊑ Receta
* Almuerzo ⊑ Receta
* Cena ⊑ Receta
* CocinaLatina, CocinaEuropea, CocinaOriental ⊑ TipoCocina

Además se utilizaron vocabularios externos:

* foaf:name para subpropiedad de tieneNombre.
* Prefijos rdfs: y rdf: para dominio y tipo.

### **3.3 Propiedades RDF**

Se definieron **10 propiedades originales** (rdf:Property), cada una con dominio y rango:

| **Propiedad** | **Dominio** | **Rango** | **Descripción** |
| --- | --- | --- | --- |
| tieneIngrediente | Receta | Ingrediente | Ingredientes de una receta |
| usaInstrumento | Receta | Instrumentos | Utensilios usados en la receta |
| usaInstrumentoEspecial | Receta | Instrumentos | Subpropiedad de usaInstrumento |
| perteneceA | Receta | TipoCocina | Tipo de cocina de la receta |
| tiempoPreparacion | Receta | xsd:integer | Tiempo en minutos para preparar |
| tieneDificultad | Receta | xsd:integer | Nivel de dificultad (1–10) |
| nivelPopularidad | Receta | xsd:integer | Popularidad (1–10) |
| momentoDia | Receta | xsd:string | Momento: Mañana, MedioDia, Tarde, Noche |
| tieneNombre | Receta | xsd:string | Nombre descriptivo (subPropertyOf foaf:name) |
| tieneInstrucciones | Receta | xsd:string | Texto con instrucciones de preparación |

Se incluyó al menos **un caso de subPropertyOf** (usaInstrumentoEspecial ⊑ usaInstrumento).

### **3.4 Instanciación de Individuos**

Se crearon **4 individuos por clase** como mínimo. Ejemplos:

* Recetas: Tacos, Pizza, Wontons, Macedonia.
* Ingredientes: Cebolla, Queso, Jengibre, FrutasVariadas.
* Instrumentos: Sartén, Horno, Olla, Batidora.
* Tipos de Cocina: CocinaItaliana, CocinaTailandesa, CocinaVegetariana, CocinaFusion.

A cada individuo se le asignaron propiedades concretas con valores literales y URI, por ejemplo:

grafo.add((name.Pizza, name.tiempoPreparacion, Literal(50, datatype=XSD.integer)))

grafo.add((name.Tacos, name.usaInstrumento, name.Sartén))

### **3.5 Razonamiento y Casos de Inferencia**

Se aplicó el razonador **RDFS Semantics** mediante:

DeductiveClosure(RDFS\_Semantics).expand(grafo)

Esto generó nuevos triples automáticos. A continuación, se documentan los **3 casos requeridos**:

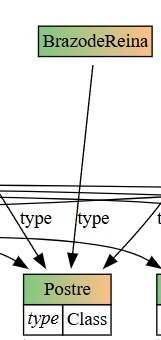
#### **Caso 1: Jerarquía de Clases**

Inferencia 1

* Antes:

  
:BrazodeReina rdf:type :Postre .

:Postre rdfs:subClassOf :Receta .



* Después de razonar, se infiere:

:BrazodeReina rdf:type :Receta .

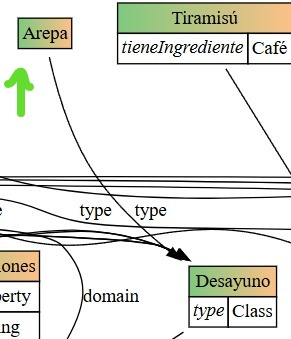
#### 

Inferencia 2

* Antes:

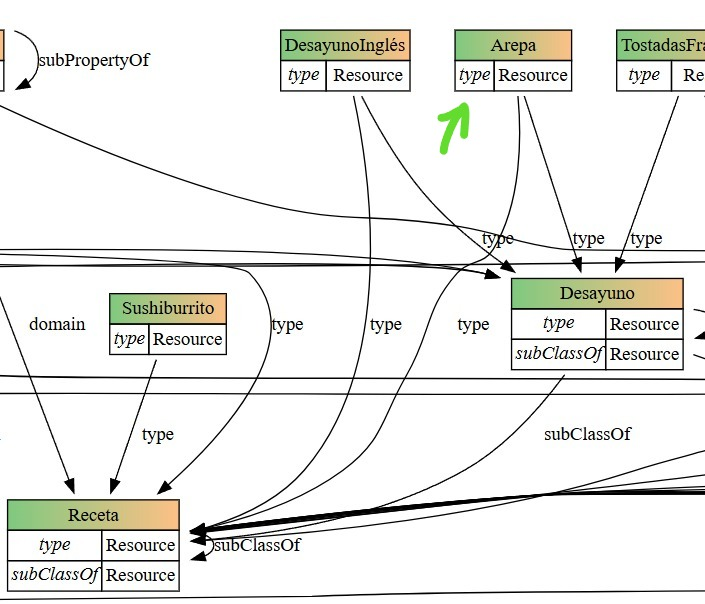
  
:Arepa rdf:type :Desayuno .

:Desayuno rdfs:subClassOf :Receta .



* Después de razonar, se infiere:

:Arepa rdf:type :Receta .



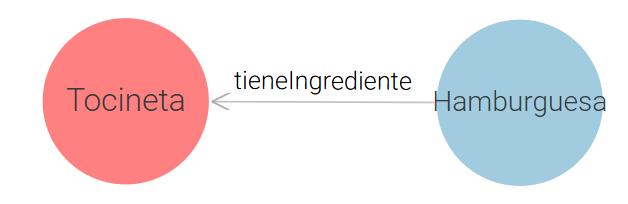
**Nota: Para este caso se usó un graficador distinto a GraphDB debido a que cuando se graficaba con GraphDB este software mostraba de forma incorrecta que los objetos tipo Desayuno, Postre, Cena y Almuerzo ya eran del tipo receta aunque esa tripleta no existiese.**

#### **Caso 2: Inferencia por Dominio y Rango**

Inferencia 1

* Antes:

:Hamburguesa ex:tieneIngrediente ex:Tocineta .



* Dado que tieneIngrediente tiene dominio Receta y rango Ingrediente, se infiere:

  
:Hamburguesa rdf:type :Receta .

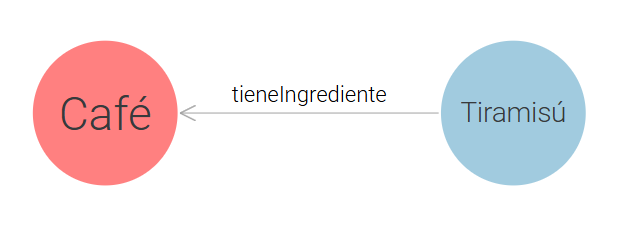
:Tocineta rdf:type :Ingrediente .

#### 

Inferencia 2

* Antes:

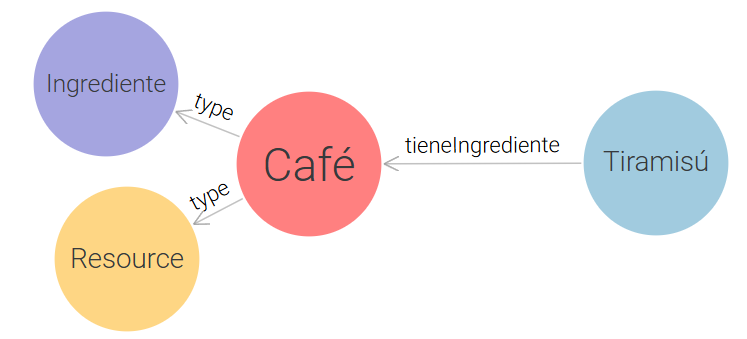
:Tiramisú ex:tieneIngrediente ex:Café .



* Dado que tieneIngrediente tiene dominio Receta y rango Ingrediente, se infiere:

  
:Tiramisú rdf:type :Receta .

:Café rdf:type :Ingrediente .



#### 

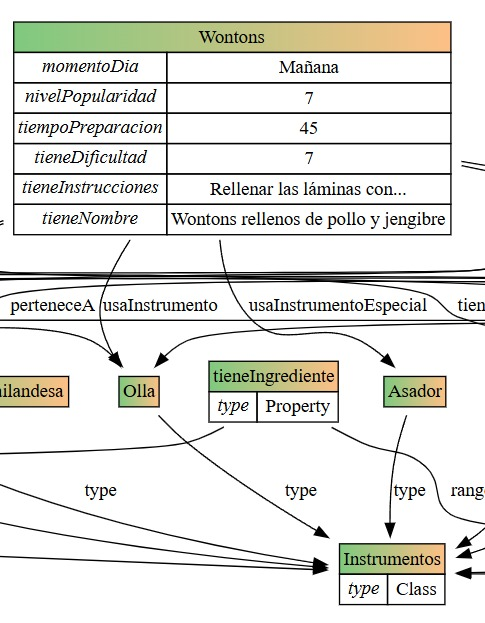
#### **Caso 3: SubPropiedades**

Inferencia 1

* Antes:

  
:usaInstrumentoEspecial rdfs:subPropertyOf :usaInstrumento .

:Wontons ex:usaInstrumentoEspecial ex:Asador .



* Después de razonar, se infiere:

:Wontons ex:usaInstrumento ex:Asador .

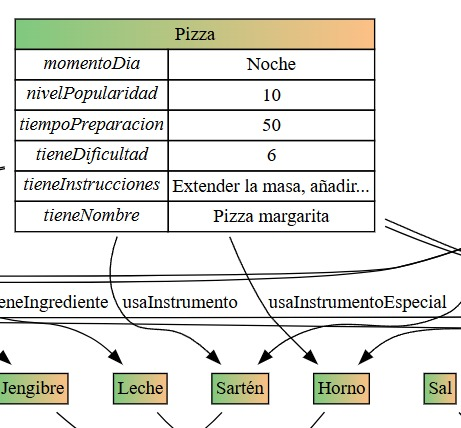
### 

### Inferencia 2

* Antes:

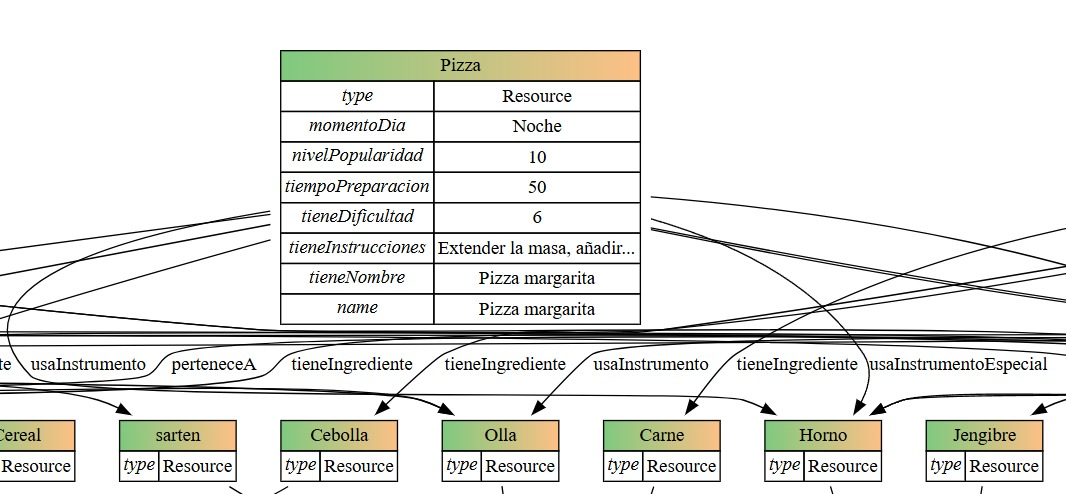
  
:usaInstrumentoEspecial rdfs:subPropertyOf :usaInstrumento .

:Pizza ex:usaInstrumentoEspecial ex:Horno .



* Después de razonar, se infiere:

:Pizza ex:usaInstrumento ex:Horno .



### **3.6 Comparación del Grafo**

* **Triples antes del razonamiento**:  
  Se registraron *327* tripletas (valor obtenido con len(grafo) antes).

Tripletas antes del razonamiento: 327

Ejemplo de tripletas en el grafo:

http://pratica1.org/recetas#tiempoPreparacion http://www.w3.org/2000/01/rdf-schema#range http://www.w3.org/2001/XMLSchema#integer

http://pratica1.org/recetas#CocinaEuropea http://www.w3.org/2000/01/rdf-schema#subClassOf http://pratica1.org/recetas#TipoCocina

http://pratica1.org/recetas#Hamburguesa http://pratica1.org/recetas#momentoDia Noche

http://pratica1.org/recetas#DesayunoInglés http://pratica1.org/recetas#usaInstrumento http://pratica1.org/recetas#Sartén

http://pratica1.org/recetas#Sushiburrito http://pratica1.org/recetas#tieneNombre Sushiburrito

http://pratica1.org/recetas#Brigadeiros http://pratica1.org/recetas#usaInstrumentoEspecial http://pratica1.org/recetas#Batidora

http://pratica1.org/recetas#Alga http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://pratica1.org/recetas#Ingrediente

http://pratica1.org/recetas#Wontons http://pratica1.org/recetas#tieneDificultad 7

http://pratica1.org/recetas#RolloDeCanela http://pratica1.org/recetas#perteneceA http://pratica1.org/recetas#CocinaEuropea

http://pratica1.org/recetas#TostadasFrancesas http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://pratica1.org/recetas#Desayuno

* **Triples después del razonamiento**:  
  Se obtuvieron *665* triples, evidenciando al menos **4 nuevos hechos** inferidos.

Tripletas después del razonamiento: 665

Algunas tripletas nuevas:

(rdflib.term.URIRef('http://pratica1.org/recetas#Horno'), rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'), rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Resource'))

(rdflib.term.URIRef('http://pratica1.org/recetas#CocinaLatina'), rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#subClassOf'), rdflib.term.URIRef('http://pratica1.org/recetas#CocinaLatina'))

(rdflib.term.URIRef('http://pratica1.org/recetas#Tacos'), rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'), rdflib.term.URIRef('http://pratica1.org/recetas#Receta'))

(rdflib.term.URIRef('http://pratica1.org/recetas#CocinaChina'), rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'), rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Resource'))

(rdflib.term.URIRef('http://pratica1.org/recetas#Masa'), rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'), rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Resource'))

(rdflib.term.URIRef('http://pratica1.org/recetas#Café'), rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'), rdflib.term.URIRef('http://pratica1.org/recetas#Ingrediente'))

(rdflib.term.URIRef('http://pratica1.org/recetas#Tacos'), rdflib.term.URIRef('http://pratica1.org/recetas#usaInstrumento'), rdflib.term.URIRef('http://pratica1.org/recetas#Horno'))

(rdflib.term.URIRef('http://pratica1.org/recetas#CocinaColombiana'), rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'), rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Resource'))

(rdflib.term.URIRef('http://pratica1.org/recetas#TostadasFrancesas'), rdflib.term.URIRef('http://xmlns.com/foaf/0.1/name'), rdflib.term.Literal('Tostadas francesas', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#string')))

(rdflib.term.URIRef('http://pratica1.org/recetas#CocinaEuropea'), rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'), rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Resource'))

Estos resultados se muestran en la consola al ejecutar el notebook.

**3.7 Diagrama de Clases de la Ontología**

Se incluye un diagrama de clases (UML) que muestra:

* Clases y sus relaciones de herencia.
* Propiedades principales y sus dominios/rangos.

## 

## **5. Sustentación**

**Pitch de sustentación (enlace al video):**[Video Pitch - Practica 1 - Equipo 2 - G2](https://youtu.be/6BHRB48YDp0)