Clase de Inteligencia Artificial:
Proyecto de Representación del Conocimiento

Profesores:
Luis A. Pineda Cortés,
IIMAS,UNAM
Ayudantes
Noé Hernández,
Ricardo Cruz,
IIMAS, UNAM

19 Septiembre, 2024

Condiciones de Entrega

Formato: El código fuente se deberá comprimir en un archivo llamado Proyecto.zip. Mientras que dentro de un archivo doc.pdf se deberá incluir la documentación y descripción de los predicados solicitados en la descripción del proyecto (indicando a que se refieren sus argumentos y mostrando por lo menos dos ejemplos de uso).

Fecha y modo de entrega: Subir el código y la documentación al classroom del curso a más tardar el 31 de octubre. El proyecto puede ser implementado en Python o Prolog. Recuerden que los equipos pueden ser de hasta 5 personas y solamente una debe subir el proyecto al classroom, agregando un README con el nombre del resto de los integrantes.

Descripción

A partir de la jerarquía conceptual con defaults y excepciones explicada en clase y de la especificación de la base de conocimiento:

1. Crear predicados para consultar:

   (a) La extensión de una clase (el conjunto de todos los objetos que pertenecen a la misma, ya sea porque se declaren directamente o porque están en la cerradura de la relación de herencia). Llevará por nombre: *class_extension*, y recibirá tres argumentos: (i) el nombre de la clase de la que se busca su extensión, (ii) la base de conocimientos en cuestión, y (iii) el resultado de la extensión en una lista.

   (b) La extensión de una propiedad (mostrar **todos los objetos** que tienen una propiedad específica ya sea por declaración directa o por herencia, incluyendo **su respectivo valor**). Llevará por nombre: *property_extension*, y recibirá tres argumentos: (i) el nombre de la propiedad de la que se busca su extensión, (ii) la base de conocimientos en cuestión, y (iii) el resultado de la extensión en una lista.

   (c) La extensión de una relación (mostrar **todos los objetos** que tienen una relación específica

ya sea por declaración directa o por herencia, incluyendo **todos los objetos** con quién están relacionados). Llevará por nombre: *relation_extension*, y recibirá tres argumentos: (i) el nombre de la relación de la que se busca su extensión, (ii) la base de conocimientos en cuestión, y (iii) el resultado de la extensión en una lista.

(d) Todas las clases a las que pertenece un objeto. Llevará por nombre: *classes_of_individual*, y recibirá tres argumentos: (i) el nombre del objeto, (ii) la base de conocimientos en cuestión, y (iii) el resultado en una lista con las clases a la que el objeto pertenece.

(e) Todas las propiedades de un objeto o clase, los predicados llevarán por nombre *properties_of_individual* y *class_properties*, respectivamente. Recibirá tres argumentos: (i) el nombre del objeto o clase, (ii) la base de conocimientos en cuestión, y (iii) el resultado en una lista con el valor de todas las propiedades del objeto o clase.

(f) Todas las relaciones de un objeto o clase, los predicados llevarán por nombre *relations_of_individual* y *class_relations*, respectivamente. Recibirá tres argumentos: (i) el nombre del objeto o clase, (ii) la base de conocimientos en cuestión, y (iii) el resultado en una lista con las relaciones del objeto/clase junto con los objetos/clases con quienes se guarda dicha relación.

2. Crear predicados para añadir:

(a) Clases u objetos cuyo nombre será *add_class* y *add_object*, respectivamente. Ambos predicados recibirán cuatro argumentos: (i) el nombre de la clase u objeto a añadir, (ii) el nombre de la clase madre, (iii) la base de conocimientos actual, y (iv) la nueva base de conocimientos donde se refleja la adición.

(b) Propiedades de clases u objetos cuyo nombre será *add_class_property* y *add_object_property*, respectivamente. Ambos predicados recibirán cinco argumentos: (i) el nombre de la clase u objeto, (ii) el nombre de la propiedad a añadir, (iii) el valor de dicha propiedad, (iv) la base de conocimientos actual, y (v) la nueva base de conocimientos donde se refleja la adición.

(c) Relaciones de clases u objetos cuyo nombre será *add_class_relation* y *add_object_relation*, respectivamente. Ambos predicados recibirán cinco argumentos: (i) el nombre de la clase u objeto, (ii) el nombre de la relación a añadir, (iii) la o las clases con quienes se guarda la relación, si se trata de *add_class_relation*, y el o los objetos con quienes se guarda la relación, si el predicado es *add_object_relation* , (iv) la base de conocimientos actual, y (v) la nueva base de conocimientos donde se refleja la adición.

3. Crear predicados para eliminar:

(a) Clases u objetos cuyo nombre será *rm_class y rm_object*, respectivamente. Ambos predicados recibirán tres argumentos: (i) el nombre de la clase u objeto a eliminar, (ii) la base de conocimientos actual, y (iii) la nueva base de conocimientos donde se refleja la eliminación.

(b) Propiedades específicas de clases u objetos cuyo nombre será *rm_class_property* y *rm_object_property*, respectivamente. Ambos predicados recibirán cuatro argumentos: (i) el nombre de la clase u objeto, (ii) el nombre de la propiedad a eliminar, (iii) la base de conocimientos actual, y (iv) la nueva base de conocimientos donde se refleja la eliminación.

(c) Relaciones específicas de clases u objetos cuyo nombre será *rm_class_relation* y *rm_object_relation*, respectivamente. Ambos predicados recibirán cuatro argumentos: (i) el nombre de la clase u objeto, (ii) el nombre de la relación a eliminar, (iii) la base de conocimientos actual, y (iv) la nueva base de conocimientos donde se refleja la

eliminación.

4. Crear predicados para modificar:

   (a) El nombre de una clase u objeto cuyo nombre será *change_class_name* y *change_object_name*, respectivamente. Ambos predicados recibirán cuatro argumentos: (i) el nombre de la clase u objeto a modificar, (ii) el nuevo nombre para dicha clase u objeto, (iii) la base de conocimientos actual, y (iv) la nueva base de conocimientos donde se refleja la modificación.

   (b) El valor de una propiedad específica de una clase u objeto cuyo nombre será *change_value_class_property* y *change_value_object_property*, respectivamente. Ambos predicados recibirán cinco argumentos: (i) el nombre de la clase u objeto, (ii) el nombre de la propiedad a modificar, (iii) el nuevo valor de dicha propiedad, (iv) la base de conocimientos actual, y (v) la nueva base de conocimientos donde se refleja la modificación.

   (c) Con quién mantiene una relación específica una clase u objeto cuyo nombre será *change_value_class_relation* y *change_value_object_relation*, respectivamente. Ambos predicados recibirán cinco argumentos: (i) el nombre de la clase u objeto, (ii) el nombre de la relación a modificar, (iii) la o las clases con quienes ahora se guarda la relación, si se trata *de change_value_class_relation*, y el o los objetos con quienes ahora se guarda la relación, si el predicado es *change_value_object_relation*, (iv) la base de conocimientos actual, y (v) la nueva base de conocimientos donde se refleja la modificación.

5. El formato de la base de conocimiento es un archivo de texto con una lista, donde los elementos de la lista son funtores de la forma:

   class(nombre_de_la_clase, clase_madre, lista_de_propiedades_de_la_clase, lista_de_relaciones_de_la_clase, lista_de_objetos)

   Tal que la lista de objetos se conforma a su vez de listas del siguiente modo:

   [id=>nombre_del_objeto, lista_de_propiedades_del_objeto, lista_de_relaciones_del_objeto]

6. Para todos los predicados deberás:

- Tomar en cuenta el mecanismo de herencia con defaults y excepciones.
- Utilizar el criterio de especificidad para resolver el problema de la no-monoticidad.
- Tomar en cuenta la interpretación de las relaciones explicada en clase.
- Cargar toda la base de conocimiento del archivo como un objeto en una sola variable (específicamente, en una lista), y utilizarla de esta manera durante toda la consulta. Al finalizar, si hubo modificaciones de la base, almacenar el resultado sobreescribiendo en el archivo del que se leyó.
- Las propiedades y las relaciones estarán en listas atributo-valor mediante el operador =>.
- Manejar información incompleta (el sistema debe responder sí, no o no sé).
- Los objetos podrán ser anónimos.
- El valor de las propiedades puede ser indeterminado.

- Respetar el nombre de los predicados, así como el orden de sus argumentos, presentando en los incisos anteriores.

Observaciones:

- Utilizar SWI Prolog versión 8.0.3 o superior o Python 3 o superior.