Universidad Veracruzana

Inteligencia Artificial

Programación para la Inteligencia Artificial

Actividad 1

Leonardo Flores Torres

13 de octubre de 2022

1. Regístrese como estudiante en la plataforma `https://www.socrative.com` para unirse a la sala (room) **2022PIA**. Resuelva el cuestionario que ahí se presenta. Las preguntas pueden tener varias respuestas **(30 puntos)**.

   Solución:

   La solución a este inciso no se incluye en el presente trabajo ya que queda registro de ello en la plataforma mencionada.

---

2. Lea el artículo K. Knight, *Unification: a multidisciplinary survey.* ACM Comput. Surv., 21(1):93-124, 1989. Resuelva los siguientes ejercicios **(50 puntos)**:

   - ¿Cuál es la diferencia entre el match y unificación? Proponga un ejemplo donde la unificación tenga éxito, pero el match no.

     Solución:

     Según Triska [1] la unificación es una generalización del match, mientras que Knight [2] menciona que el match es una variante de la unificación. Lo cierto es que la unificación y el match son similares, pero no son lo mismo. Knight ejemplifica la unificación con dos términos t_1 y t_2, estos son unificables si existe una sustitución θ tal que $\theta(t_1) = \theta(t_2)$. Mientras que el match se da en el caso que exista una sustitución θ' tal que $\theta'(t_1) = t_2$. A pesar de contar con las definiciones formales de lo que es unificación y match, considero conveniente mencionar un par de las reglas generales para realizar match mencionadas por Bratko [3],

     a) Si t_1 y t_2 son constantes entonces ambos hacen match si y sólo si son el mismo objeto.

     b) Si t_1 es una variable y t_2 algo distinto a una variable entonces hay match.

     Retomando la definición formal entre la unificación y el match, se puede observar que la unificación es bidireccional. Por el contrario, el match es unidireccional [2].

     A continuación se muestra un ejemplo de lo anteriormente mencionado. Supóngase que al programa de la familia se agrega un hecho `progenitor(bob, roger).`, entonces `ann` y `pat` tendrían un hermano. Esta regla se puede definir como:

```
% La regla hermano donde X es hermano de Y sin distinción de su sexo
↪  biológico.
% hermanoneutro/2
hermanoneutro(X, Y).
```

No es necesario usar un programa para este caso pero viene bien para mostrar el ejemplo ¿Que pasa si preguntamos de quien es hermano `ann`, y quién es hermano de `ann` independientemente de su sexo biológico?

```
?- hermanoneutro(ann, X) = hermanoneutro(X, ann).
X = ann.
```

Es interesante ver que se llevó a cabo una unificación tal que $\theta = \{X \backslash ann\}$. Pero si se aplicase la sustitución θ solamente a `hermanoneutro(ann, X)` no habría match con el término del lado derecho.

- Ejemplifique un caso en que omitir el chequeo de ocurrencias, hace que Prolog de una respuesta errónea al computar una meta.

  Solución:

  `SWI-Prolog` no hace chequeo de ocurrencias, lo que hace es tratar de unificar una variable con un término que ya contiene a esa misma variable. Un caso en el que sucede esto se puede observar en el ejemplo propuesto por Knight [2]:

```
% Todo número es menor que su sucesor.
less(X, s(X)).

% Se computa verdadero o falso si hay algun número que sea mayor que su
↪  sucesor.
foo :- less(s(Y), Y).
```

  Al importar el programa en `SWI-Prolog` y llamar `foo` da un resultado afirmativo a la pregunta ¿existe algún número mayor a su sucesor?

```
?- foo.
true.
```

- Describa una aplicación de la unificación además de la programación lógica.

  Solución:

  Al final del artículo se mencionan 3 áreas donde la unificación también es usada aparte del área de la programación lógica como lo son en la inferencia de tipos, en lenguajes de programación para reconocimiento de patrones, y en aprendizaje automático.

El reconocimiento de patrones es algo común implementado en alguna manera en los lenguajes de programación populares hoy en día. Por ejemplo, `julia`, `python` y `rust` tienen alguna forma de pattern-matching dentro de sus librerias base, aunque tambien hay librerias en esos lenguajes de programación mas especializadas para realizar dicha tarea. El reconocimiento de patrones es conveniente cuando se desea hacer algun tipo de preprocesamiento de datos en archivos de texto pero no es la única aplicación. Dos lenguajes de programación no tan populares pero si usados ampliamente son `erlang` y `awk`. Ambos hacen uso del reconocimiento de patrones, de hecho `awk` fue creado con ese propósito principal en mente para procesar patrones en archivos de texto.

Otra de las aplicaciones es en el area de la inferencia de tipos durante el tiempo de compilado. Uno de los primeros algoritmos para la inferencia de tipos fue hecho por Milner haciendo uso de la unificación [2]. El lenguaje de programación `c++` puede usar una palabra clave `auto` para dejar que el compilador elija el tipo de una variable de manera conveniente. Otro lenguaje de programación compilado es `rust` aunque este no permite inferencia de tipos automática, el usuario tiene que definir de antemano los tipos de sus variables, del retorno de sus funciones y de sus estructuras. Esto era una práctica común en los lenguajes de programación de antaño.

Un lenguaje de programación podría ejemplificar, en mayor o menor medida, lo que se puede hacer con inferencia de tipos. `julia` es un lenguaje multiparadigma *compilado justo a tiempo*, a comparación de `rust` donde los tipos de variables o funciones definidos a priori son inmutables, en `julia` es posible definir o no definir los tipos a priori. Además, su compilador es lo bastante capaz de inferir que tipos son los más adecuados y puede realizar promoción de tipos si variables de distintos tipos interactúan. En cambio `rust` no realiza ninguna promoción de tipos, el usuario tendría que hacerlo manualmente si así lo necesitase.

- De las propiedades de la unificación resumidas al final del artículo, ilustre con ejemplos dos de ellas.

  Solución:

  Las propiedades a ejemplificar se enlistan a continuación:

  a) La unificación es un proceso de reconocimiento de patrones (*pattern-matching*), esta se encarga de determinar si dos estructuras hacen match usando el mecanismo de sustitución de variables. Esto se puede observar en una estructura en particular que se ha venido usando en el curso, las listas.

```
?- [X | Xs] = [1, 2, 3, 4].
X = 1,
Xs = [2, 3, 4].
```

  Se reconoce el patrón de la lista [4] donde X es la cabeza de una lista y Xs la cola de la lista. La unificación para el primer elemento de la lista es $\theta_1 = \{X \backslash 1\}$,

y $\theta_2 = \{Xs \backslash [1, 3, 4]\}$ para la cola. Otro ejemplo similar es el siguiente:

```
?- [Y] = [X | Xs].
Y = X,
Xs = [].
```

Aunque aquí la unificación se da entre la cabeza de las listas $\theta_1 = \{X \backslash Y\}$, y la lista vacía de una con la cola de la otra lista $\theta_2 = \{Xs \backslash [\,]\}$.

b) La unificación es conmutativa y asociativa. Esto quiere decir que el orden de las unificaciones no afecta al resultado final. Por ejemplo:

```
% Pequeño programa para encontrar vecinos.
% vivencerca/2
vivencerca(juan, maria).
vivencerca(gloria, estefan).
vivencerca(maria, gloria).

% vecinos/2
vecinos(X, Y) :-
    dif(X, Y),
    vivencerca(X, Y).
vecinos(X, Y) :-
    vivencerca(Y, X),
    dif(X, Y).
```

Ambas reglas darán el mismo resultado a pesar de que las unificación en cada una de ellas se de en orden distinto, respectivamente.

```
?- vecinos(maria, X).
X = gloria ;
X = juan.

?- vecinos(X, maria).
X = juan ;
X = gloria.
```

El ejemplo anterior es una ejemplo un poco más elaborado de lo que debría para ejemplificar esta solución, pero puede verse de una manera más resumida como:

```
?- t(a, X) = t(X, a).
X = a.

?- t(X, a) = t(a, X).
X = a.
```

Que también cumple con la propiedad de conmutatividad.

c) La unificación es monótona. La unificación puede añadir información, por ejemplo, cuando una variable unifica con una constante. Pero nunca se pierde o quita

información a causa del proceso de unificación. Basándose en el mismo ejemplo de los vecinos definido anteriormente se puede ejemplificar este punto.

```
?- X = gloria, vecinos(X, estefan) = vecinos(X, Y).
X = gloria,
Y = estefan.

?- X = milagros, vecinos(juan, X) = vecinos(Y, X).
X = milagros,
Y = juan.
```

En el primer caso ya sabiamos que `gloria` y `estefan` son vecinos porque vecinos son aquellas personas que vivan cerca, esto se definió en el hecho `vivencerca(gloria, estefan)`. Pero en el segundo caso no hay nadie en nuestro universo de discurso llamada `milagros`, pero se dá la unificación y la información de que $\theta = \{X \backslash milagros\}$ se guarda.

- Califique de 1 a 10 los siguientes aspectos del artículo: relevancia del tema, calidad técnica, redacción ¿Recomendaría el artículo a alguien que estudia IA? Justifique brevemente su respuesta ¿Qué mejoras sugeriría al autor del artículo?

  a) Relevancia del tema: 10. El tema es completamente relevante a pesar de su fecha de publicación. Podrá ser el caso en otras ciencias que los avances dejan obsoletos los descrubrimientos pasados, pero este no es el caso. Todos los temas que aquí se exponen representan bases y áreas en los que la unificación llega a a la fecha a pesar de nuevos avances que hayan surgido. Personalmente podría estar sesgado ya que un área de interés propia, también mencionada en el texto, es *proof automation* y esperaría poder estudiar más de ello en un futuro próximo.

  b) Calidad técnica: 8. La calidad técnica es buena pero hace falta incluir ejemplos para apoyar a las explicaciones del autor y evitar así caer en ambiguedades. La notación está definidia al inicio del artículo pero es cierto que en ocasiones resulta confusa.

  c) Redacción: 9. El autor comenzó con un estilo de redacción poco ortodoxo refiriéndose a él mismo en el primer párrafo, lo que ya no sucedió posteriormente en el texto. Lo que podría verse como mala práctica pero no considero que sea un punto grave ya que el resto del artículo artícula bien sus ideas a pesar de faltar elaborarlas en algunos puntos importantes, como cuando es idóneo ejemplificar lo mencionado.

  d) Si recomendaría la lectura de este artículo a alguien que esté estudiando Inteligencia Artificial pero no como una lectura de naturaleza introductoria. Considero útil que el estudiante haya pasado por un periodo de acercamiento a las bases de los temas que se mencionan antes de leer este artículo.

  e) Personalmente hubiese preferido que el autor añadiera más ejemplos o ejercicios cuando expone la teoría que en este artículo se incluye. Queda en claro al inicio del artículo que el propósito de este no es este ejemplificar la teoría pero el de mostrar el panorama donde la unificación se encuentra presente. Sin embargo, nunca está de más añadir

detalles que permitan al lector comprender lo que el autor desea transmitir, mejor esto a quedar en la ambiguedad.

3. Extienda el programa de la familia en Prolog para incluir las relaciones `tio/2`, y `tia/2`. Pruébelas con las metas:
   - `tio(X, Y).`
   - `tia(ann, Y).`

   Defina una meta para computar quienes son los sobrinos en esa familia **(10 puntos)**.

   Solución:

   Para extender el programa de la familia decidí modificar la manera en como se había definido anterirmente durante las clases [5] la regla `hermano/2`, se creó una nueva regla `hermanoneutro/2` que funciona como una generalización independiente del sexo de nacimiento de las personas implicadas. Algo similar se hizo para definir la reglas `tio/2`, `tia/2` y, `sobrino/2` y `sobrina/2`.

   El extracto del programa a continuación es el programa de la familia similar a como se tenía inicialmente:

```
%%% Autor: Leonardo Flores Torres
%%% Curso: Programacion para la Inteligencia Artificial
%%% Profesor: Alejandro Guerra Hernández
%%% Version extendida del programa de la familia

%%% progenitor/2 denota el hecho sobre quién es progenitor de que persona. Este
↪  hecho no considera sexos de nacimiento, por lo que puede considerarse como
↪  neutro y usarse posteriormente en casos más específicos.
progenitor(pam, bob).
progenitor(tom, bob).
progenitor(tom, liz).
progenitor(bob, ann).
progenitor(bob, pat).
progenitor(pat, jim).

%%% mujer/1 denota el hecho de los individuos que nacieron con sexo femenino.
mujer(pam).
mujer(pat).
mujer(liz).
mujer(ann).

%%% hombre/1 denota el hecho de los individuos que nacieron con sexo masculino.
hombre(tom).
hombre(bob).
hombre(jim).
```

   Se agregaron 3 reglas, `hermanoneutro/2`, `hermano/2` y `hermana/2`. Esto se hizo para tener una mayor facilidad al tratar con los sexos de nacimiento de los individuos. A mi parecer, tener una definición de una regla de una manera más general que pueda hacerse más específica dependiendo del caso es más útil y proporciona mayor flexibilidad al extender un programa.

```
%%% hermanoneutro/2 computa quienes dos personas son hermanos, donde X es hermano de Y independientemente de su sexo biológico.
%%% Ejemplos:
%%% ?- hermanoneutro(ann, X).
%%% X = pat.
%%% ?- hermanoneutro(X, tom).
%%% false.
hermanoneutro(X, Y) :-
    dif(X, Y),
    progenitor(Z, X),
    progenitor(Z, Y).

%%% hermana/2 es un caso específico de hermanoneutro/2 donde X es femenino, y además es hermana de Y.
hermana(X, Y) :-
    mujer(X),
    hermanoneutro(X, Y).

%%% hermana/2 es un caso específico de hermanoneutro/2 donde X es masculino, y además es hermano de Y.
hermano(X, Y) :-
    hombre(X),
    hermanoneutro(X, Y).
```

Posteriormente, también se agregaron las reglas para `padre/2` y `madre/2` como se había visto en clase:

```
%%% padre/2 es una regla donde se especifíca el sexo de nacimiento de X, y además es padre de Y.
padre(X, Y) :-
    hombre(X),
    progenitor(X, Y).

%%% madre/2 es una regla donde se especifíca el sexo de nacimiento de X, y además es madre de Y.
madre(X, Y) :-
    mujer(X),
    progenitor(X, Y).
```

El mismo razonamiento se utilizó para implementar la regla `tio/2` y `tia/2`, un caso general `tioneutro/2` que se hace específico dependiendo del caso:

```
%%% tioneutro/2 computa quien es tio de Y, donde X es el tio independiente de su sexo biológico.
%%% Ejemplos:
%%% ?- tioneutro(X, jim).
%%% X = ann ;
%%% false.
%%% ?- tioneutro(jim, X).
%%% false.
tioneutro(X, Y) :-
    hermanoneutro(X, Z),
```

```
62      progenitor(Z, Y).
63
64  %%% tio/2 computa quien es tio de Y, donde X es el tio y debe tener sexo
    ↪  masculino de nacimiento.
65  tio(X, Y) :-
66      hombre(X),
67      tioneutro(X, Y).
68
69  %%% tia/2 computa quien es tia de Y, donde X es la tia y debe tener sexo femenino
    ↪  de nacimiento.
70  tia(X, Y) :-
71      mujer(X),
72      tioneutro(X, Y).
```

Finalmente, se implementó la regla general `sobrinoneutro/2` a partir de la regla `tioneutro/2`:

```
73  %%% sobrinoneutro/2 computa X sobrino de Y siempre y cuando Y sea tio de X, donde
    ↪  X es independiente de su sexo biológico y también lo es Y.
74  %%% Ejemplos:
75  %%% ?- sobrinoneutro(jim, X).
76  %%% X = ann ;
77  %%% false.
78  %%% ?- sobrinoneutro(X, ann).
79  %%% X = jim.
80  %%% ?- sobrinoneutro(X, liz).
81  %%% X = ann ;
82  %%% X = pat ;
83  %%% false.
84  sobrinoneutro(X, Y) :-
85      tioneutro(Y, X).
86
87  %%% sobrino/2 computa de quien es sobrino X, donde X es el sobrino y debe tener
    ↪  sexo masculino de nacimiento, pero no importa el sexo de quien es tio.
88  sobrino(X, Y) :-
89      hombre(X),
90      sobrinoneutro(X, Y).
91
92  %%% sobrina/2 computa de quien es sobrina X, donde X es la sobrina y debe tener
    ↪  sexo femenino de nacimiento, pero no importa el sexo de quien es tio.
93  sobrina(X, Y) :-
94      mujer(X),
95      sobrinoneutro(X, Y).
```

Lo interesante es que con estas definiciones generales es sencillo definir una regla para encontrar quienes son todos los sobrinos dentro de una familia:

```
96  %%% sobrinos/1 computa quienes son los sobrinos dentro de la familia
    ↪  independiente del sexo de nacimiento, sin importar de quien sean sobrinos.
97  %%% Ejemplo:
98  %%% ?- sobrinos(X).
99  %%% X = ann ;
```

```
%%% X = pat ;
%%% X = jim ;
%%% false.
sobrinos(X) :-
    sobrinoneutro(X, _).
```

Si se computan las metas `tio(X, Y)`, `tia(ann, Y)`, `sobrinos(X)` se obtiene lo siguiente:

```
?- tio(X, Y).
false.

?- tia(ann, Y).
Y = jim.

?- sobrinos(X).
X = ann ;
X = pat ;
X = jim ;
false.
```

---

4. Aplique el algoritmo de unificación visto en clase a los términos $f(a,\ g(Z),\ Z)$ y $f(X,\ g(X),\ b)$ **(10 puntos)**.

   Solución:

   Aplicando el algoritmo de unificación se llega a lo siguiente:

$$\begin{aligned}\{f(a,\ g(Z),\ Z) = f(X,\ g(X),\ b)\} \Rightarrow &\quad \{X = a,\ g(Z) = g(X),\ Z = b\}\\ \Rightarrow &\quad \{X = a,\ g(Z) = g(a),\ Z = b\}\\ \Rightarrow &\quad \{X = a,\ Z = a,\ Z = b\}\\ \Rightarrow &\quad \{X = a,\ Z = a,\ a = b\}\\ \Rightarrow &\quad \text{fallo}\end{aligned}$$

   Por lo tanto, el conjunto $\{f(a,\ g(Z),\ Z) = f(X,\ g(X),\ b)\}$ no tiene forma resuelta [5]. Es interesante este problema ya que se podria haber elegido aplicar el algoritmo de unificación a $\{f(X,\ g(X),\ b) = f(a,\ g(Z),\ Z)\}$ y se llegaría al mismo resultado gracias a la propiedad conmutativa.

Referencias

[1] Marcus Triska. The power of prolog. `https://www.metalevel.at/prolog`, 2005. Visitado: 2022-09-11.

[2] Kevin Knight. Unification: A multidisciplinary survey. *ACM Computing Surveys (CSUR)*, 21(1):93–124, 1989.

[3] Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2012.

[4] Lutz Hamel. An introduction to artificial intelligence with ai game development. https://homepage.cs.uri.edu/faculty/hamel/courses/2015/spring2015/csc481/lecture-notes/, 2015. Visitado: 2022-10-08.

[5] Alejandro Guerra-Hernandez. Programación para la inteligencia artificial. https://www.uv.mx/personal/aguerra/pia/, 2022. Visitado: 2022-10-07.