Implementación del algoritmo de unificación en LISP

Fundamentos de Sistemas Inteligentes

Alexandra Vicente Sánchez, Diego Martín Arroyo

El algoritmo de unificación encuentra una asignación de variables que haga idénticos dos predicados dados, si existe. El resultado del algoritmo se denomina *unificador*, y consiste en una lista de pares que indican la sustitución a realizar para cada elemento de los predicados, a fin de unificarlos.

Existen varios unificadores distintos para un mismo par de predicados, por ejemplo:

PADRE(x, HERMANO(y))
PADRE(JUAN, z)

Tiene los unificadores {JUAN/x, hermano(y)/z} y {JUAN/y, HERMANO(PEDRO)/z, PEDRO/y}. Si bien el segundo de ellos incluye una asignación innecesaria (no es necesario unificarla). Por ello siempre existe un unificador que reduce al mínimo el número de sustituciones, conocido como *unificador más general*.

También hay que tener en cuenta que la aplicación sucesiva de dos sustituciones es equivalente a la aplicación de la composición de estas sustituciones.

La implementación del algoritmo realizada tiene como resultado el unificador más general o un mensaje de error en caso de que los predicados no sean unificables. Se ha realizado la misma en LISP, y aprovecha de forma extensiva la recursividad para explorar los predicados. El pseudocódigo propuesto para el algoritmo es el siguiente:

IF alguno de ellos es un átomo, intercambiarlos si es necesario para que E1 sea un átomo y entonces **DO**

BEGIN

IF E1 = E2 **RETURN** NADA

IF E1 es variable DO

BEGIN

IF E1 aparece en E2 **RETURN** FALLO

RETURN E2/E1

END

IF E2 es variable **RETURN** E1/E2

END

F1 ← **PRIMERO**(E1); T1← **RESTO**(E1)

F2 ← **PRIMERO**(E2); T2← **RESTO**(E2)

Z1 ← **UNIFICAR**(F1, F2)

IF Z1 = FALLO **RETURN** FALLO

G1 ←**APLICAR** Z1 a T1

G2 ← **APLICAR** Z1 a T2

Z2 ← **UNIFICAR**(G1, G2)

IF G2 = FALLO **RETURN** FALLO

RETURN COMPOSICIÓN(Z1, Z2).

Aprovechando las características de LISP, hemos realizado varias modificaciones al pseudocódigo que si bien no afectan a la estructura general del algoritmo, simplifican el control de flujo.

Se han seguido las siguientes convenciones para implementar el pseudocódigo en LISP, aprovechando los elementos comúnmente utilizados en el lenguaje:

- Un predicado siempre es una lista de elementos, siendo el primero de ellos el nombre del predicado.
  Ejemplo: El predicado *P(JUAN)*, se escribirá en LISP como **(P JUAN).**
- Los elementos a evaluar serán de 3 tipos: constante (representado como un átomo en LISP, variable (representado como una lista de dos elementos, siendo el primero el símbolo **?** y el segundo el valor del elemento) y función, que es equivalente al predicado inicial.
- Un unificador es una lista que contiene a su vez listas formadas por dos elementos, siendo el primero el valor por el que se reemplaza y el segundo el elemento que es reemplazado.

El punto de partida es la función (**unificacion e1 e2)**, que verifica que el nombre de los dos predicados (**e1** y **e2**) es idéntico, y captura las excepciones que el algoritmo genera en caso de que no se haya podido realizar la unificación. Utilizando la función **let** definimos una serie de variables (en nuestro caso, únicamente el unificador **u**) y llamamos a la función **unificar**, que será la que ejecute el algoritmo de unificación.

En **unificar** contamos con 6 condiciones, en función de los parámetros de entrada **e1** y **e2**, que pueden tener como valor un único elemento (variable o constante), un conjunto de elementos o valor nulo.

- El primer condicional comprueba si ambos elementos son iguales, y marca el fin de la recursividad (es la única forma de terminar el algoritmo, excepto si se lanza alguna excepción). Se cumple cuando ambos elementos son

iguales, tanto si tienen un valor o si ambos son nulos (esto ocurre cuando se han 'consumido' todos los elementos de ambos predicados).

- La segunda condición comprueba si alguno de los dos valores es nulo, y lanza una excepción en caso de que fuera así. Si ambos son nulos serían iguales y por tanto el camino seguido sería el descrito anteriormente. Si este código es evaluado positivamente, significa que en uno de los predicados todos los elementos han sido consumidos mientras que en el otro aún quedan. Dado que siempre se procesa un elemento de cada predicado a la vez, esto significa que los predicados tienen longitudes distintas, y por tanto, no son unificables.

- Las condiciones tercera y cuarta comprueban si **e1** o **e2** son átomos o variables, tal y como se realiza en las líneas 4 a 9 del pseudocódigo. Si alguna de las condiciones se satisface, se añaden al unificador.

- La quinta condición verifica que **e1** o **e2** son átomos (constantes). Si alguno de los dos es variable, el algoritmo hubiera continuado por las condiciones tercera o cuarta, lo cual significa que ninguno de ellos es variable, y por tanto no es posible generar un unificador (no se pueden unificar átomos). Se lanza una excepción que indica que el algoritmo no es unificable.

- Si ninguna de las anteriores condiciones es satisfecha, **e1** y **e2** son conjuntos de elementos, y por tanto se procederá a procesarlos tal y como se define en las líneas 12 a 19 del pseudocódigo, con varias modificaciones:

Tras definir los valores de **f1**, **f2**, **t1**, **t2** (tal y como se describe en el pseudocódigo), llamamos recursivamente a **unificar** con el valor de **f1** y **f2** tras aplicar sobre este los elementos ya añadidos al unificador. El resultado de esta llamada recursiva constituirá el nuevo valor del unificador.

Posteriormente llamaremos a la función **unificar** con los valores de **t1** y **t2**, siendo el valor de retorno de esta función el valor de retorno de la invocación superior.

La composición de los unificadores se realiza de forma implícita al añadir nuevos elementos al unificador. En la función **annadir** se llama a la función **hacer-sustitucion**, que se encarga de aplicar el nuevo par añadido al resto de elementos, siendo el resultado el mismo que el de realizar la operación composición.

Se define la función **sustituir** para realizar las diferentes sustituciones de elementos en un unificador. **hacer-sustitucion** invoca a esta función tras realizar un conjunto de operaciones preliminares, y también se utiliza de forma directa en **annadir**. Es una función recursiva, que recorre todos los elementos de la lista y analiza si es necesario realizar una sustitución o no.

Se han creado varias funciones auxiliares:

- Se define la función **esvariable** para evaluar si un elemento es de tipo variable o no.

- Se define la función **extraerSimbolo** para abstraer la comparación de símbolos. Retorna siempre el valor de un elemento, sea constante o variable, y de esta forma, poder comprobar valores sin contar con varios casos para cada tipo de dato.

Debido a la naturaleza recursiva del algoritmo, los primeros elementos en ser añadidos serán los nodos hoja, por lo que el unificador resultante estará en orden inverso. Utilizamos la función **reverse** para cambiar los elementos al orden deseado.

Pruebas unitarias

En el archivo test.lisp se observa la batería de pruebas realizada sobre cada una de las partes que conforman la implementación del algoritmo, evaluando todos los tipos de datos que pueden ser datos como entrada al algoritmo, y la salida que estos producen.

Bibliografía consultada:

Barski, C. 2010. Land of Lisp: Learn to Program in Lisp, One Game at a Time!
GNU Emacs LISP Reference Manual (consultado el 23 de octubre de 2014): https://www.gnu.org/software/emacs/manual/html_node/elisp/index.html

Un algoritmo de Unificación (consultado el 15 de noviembre de 2014): http://www.davioth.com/pl/node201.html
Apuntes de inteligencia Artificial (consultado el 23 de diciembre de 2014): http://users.dcc.uchile.cl/~abassi/Cursos/41a/unificacion.html
Ejemplo de **Luger G., Stubblefield, W. 2008** Artificial Intelligence: Structures and strategies for complex problem solving
http://www.cs.unm.edu/~luger/ai-final/code/LISP.unification.html
Ejemplo de **Norvig, P. 1991** Paradigms of Artificial Intelligence Programming
http://norvig.com/paip/unify.lisp
Ejemplo de **Tanimoto, S.** A Unification algorithm for literals in the Predicate Calculus
http://web.cecs.pdx.edu/~mperkows/CLASS_ROBOTICS/LISP/tanimoto/UNIFY.CL