Inteligencia Artificial II

Algoritmo de unificación

Procesamiento de Lenguaje Natural

Rocío Romero Moreno
Abel Sayago Galván

Índice

1. Introducción
2. Estructura del Proyecto
    2.1 Estructuras de Datos Auxiliares
        2.1.1 Rasgos
        2.1.2 Conjunto de Rasgos
        2.1.3 ECR (Estructura Compleja de Rasgos)
        2.1.4 Par
    2.2 Parser
    2.3 Unificador
    2.4 Entorno
3. Ejecuciones Realizadas
4. Problemas Surgidos
5. Cómo utilizar el Algoritmo

Introducción

El objetivo de este trabajo es implementar un sistema que permita representar estructuras complejas de rasgos unificadas mediante un algoritmo de unificación.

Una estructura de rasgos se define como una función parcial desde un conjunto de atributos en un conjunto de valores. Se pueden representar mediante arrays, de la siguiente forma

$$\lambda_2 = \begin{bmatrix} head & comer \\ agr & \begin{bmatrix} num & sing \\ pers & 3 \end{bmatrix} \\ tense & past \end{bmatrix}$$

Se considera que una estructura de rasgos subsume a otra si cada elemento de información de la primera está incluido en la segunda.

Dadas dos estructuras, la unificación de estas será una nueva estructura de rasgos, del tal manera que la resultante es la menor estructura subsumida por P y Q. Por ejemplo:

Teniendo las estructuras siguientes:

$$\alpha_1 = \begin{bmatrix} genero & masculino \\ numero & singular \end{bmatrix}$$

$$\alpha_3 = \begin{bmatrix} genero & masculino \\ persona & 2 \end{bmatrix}$$

La estructura de rasgos resultante sería:

$$\alpha_2 = \begin{bmatrix} genero & masculino \\ persona & 2 \\ numero & singular \end{bmatrix}$$

Nuestro trabajo consiste en implementar un algoritmo de unificación. Los archivos de entrada son dos:

- Un archivo **.ecr** , al que podemos llamar diccionario, que contiene las estructuras de rasgos definidas.
- Por otro lado, tenemos un archivo **.unf** , que tiene llamadas al algoritmo de la siguiente manera:

```
$Unifica(ecr1&ecr2)
```

Devolverá la unificación de las distintas llamadas al algoritmo, en un archivo **.trace**, en la ubicación y con el nombre que deseemos. En caso de no ser unificables, el algoritmo imprimiría en el archivo de salida que no ha sido posible aplicar el algoritmo.

El lenguaje que hemos escogido para realizar la implementación, ha sido Java, y el entorno de programación Eclipse 3.6.

Estructuras de datos auxiliares

Rasgos

Definimos un Rasgo, como una clase que tiene como atributos:

- Un *String* llamado *key*, que representa el nombre del rasgo a definir
- Y un *T value* , que puede ser tanto un *String* como un *Conjunto de Rasgos* que definimos más adelante. Es el valor del rasgo cuya etiqueta es *key*.

Conjunto de Rasgos

Un conjunto de rasgos, es una clase que tiene un solo atributo:

- Un *HashSet<Rasgos>* llamado *rasgos* , que es un set de Rasgos.

ECR

La clase de las Estructuras Complejas de Rasgos, está definida por dos atributos:

- Un *String* llamado *etiqueta*, que representa el nombre de la estructura que estamos definiendo.
- Y un Set de *Conjuntos de Rasgos*, llamado *conjuntosRasgos*

Lo implementamos de esta manera, para que a la hora de almacenar aquellas estructuras que tuviesen más de una definición (misma etiqueta, distintos rasgos) se guardasen todas en la misma ECR, de tal forma que si una etiqueta aparece una sola vez en el diccionario, el tamaño de *conjuntosRasgos* sería uno y si apareciese más de una vez la misma etiqueta, el tamaño sería igual al número de veces que apareciese dicha etiqueta.

Par

Es una estructura auxiliar. Es simplemente una tupla, con dos posiciones. La creamos para facilitar las devoluciones de los métodos y pasar datos entre los distintos métodos.

Parser

El parser tiene como misión obtener las estructuras a unificar a partir del **.unf** y averiguar cómo son estas en el diccionario (**.ecr**). Tenemos dos métodos principales:

- Lectura del **.unf** : Nuestro parser comienza analizando el **.unf** . El método recibe como parámetro de entrada la ruta al fichero **.unf**, lo abre y comienza a leerlo. Al final, devuelve dos estructuras, una lista ordenada de pares de String, que son aquellos que debemos unificar, y un Set de String, aquellos que debemos buscar en el diccionario.

Una vez obtenidas, pasaríamos a analizar el archivo **.ecr**

- Lectura **.ecr**: para este proceso probamos con varias alternativas, primero intentamos leer todo el archivo y crear todas las estructuras posibles, pero la máquina virtual de java se quedaba sin memoria rápidamente, así que optamos por crear únicamente las estructuras que fueran necesarias, es decir, aquellas que aparecieran en el

archivo **.unf**, funcionaba mucho mejor, bastante más rápido, pero en los test más grandes también se quedaba sin memoria, así que optamos por una tercera y última opción: no guardar las estructuras, únicamente las cadenas que las representan y crear las estructuras en el momento de realizar la unificación. Así, este método recibe la ruta al fichero**.ecr** y un Set de String que contiene las etiquetas que estamos buscando y devuelve un Map que tiene por clave la etiqueta del ECR y por valor un Set de String, uno por cada definición del ECR aparece en el diccionario.

Debido a la forma en la que decidimos realizar la lectura del .**ecr** necesitamos otro método que dado un String que representa a un conjunto de rasgos, nos devolviera un objeto de tipo ConjuntoRasgos, que será el que posteriormente añadamos al ECR.

Este método son en realidad dos, se trata de una doble recursión y funcionan, a groso modo, como sigue:

- Parsear el conjunto de rasgos: divide la cadena por las comas que separa a cada rasgo dentro del conjunto y con cada uno llama al parser del rasgo, se añaden todos los rasgos al conjunto y se devuelve

-Parsear el rasgo: divide la cadena en dos, separándola por el primero carácter ':', la primera parte de la cadena será la etiqueta y la segunda parte, si el primer carácter es un '(' será un conjunto de rasgos, así que se llama al método anterior para que nos devuelva el conjunto y si no es un carácter '(', entonces es un String, se crea el rasgos y se devuelve.

Como con el resto de detalles de implementación, todo está extensamente explicado en el código de la clase Parser.

Unificador

El Unificador es la clase principal de todo el programa, es donde se encuentra el algoritmo que hace la unificación y el que escribe la salida de cada una de las unificaciones.

El algoritmo que realiza la unificación se divide en dos, por un lado un método que dados dos conjuntos de rasgos los unifica (si es posible) en otro y un segundo método que dado dos ECR devuelve un String formateado con lo que vamos a escribir en el fichero de salida: las entradas del algoritmo y su salida.

- ResultadoUnificarECR: recibe dos ECR, guarda una cadena con los dos ECR de la entrada y por cada uno de los conjuntos de rasgos que tiene el primer ECR realiza la unificación con cada uno de los conjuntos de rasgos del segundo ECR y escribe el resultado en la cadena, que devuelve como parámetro de salida.

- Unifica: recibe dos conjuntos de rasgos y devuelve un tercer conjunto de rasgos resultado de la unificación para ello, primero se copian los conjuntos de rasgos (ya que se van a ir eliminando sus componentes), posteriormente por cada rasgo de uno de los conjuntos de rasgos se recorre el segundo hasta encontrar un rasgo que tenga la misma etiqueta, si es así y son de tipo cadena, se comparan sus valores, si coinciden se añade el rasgo al conjunto de rasgos que se va a devolver, si tiene por valor un conjunto de rasgos, entonces se realiza la unificación entre los valores de uno y otro rasgo, si la unificación se realiza correctamente, se crea un nuevo rasgos con la misma etiqueta y por valor el resultado de la unificación y se añade al conjunto que se va a devolver, en cualquier caso, se elimina el rasgo del segundo conjunto (para mejorar el tiempo del algoritmo). En caso de que no encontremos un rasgo con la misma etiqueta en el segundo conjunto, agregamos el rasgo al resultado. Al finalizar el bucle, añadimos todos los rasgos que queden en b al conjunto de salida y lo devolvemos.
Aquí tenemos una versión de lo anteriormente explicado en pseudocódigo:

```
- unifica(ConjuntoRasgos a, ConjuntoRasgos b) : ConjuntoRasgos
        copiamos a y b en ca y cb
        mientras queden elementos en ca, sea ra el siguiente elemento de ca
                mientras queden elementos en cb y no lo hayamos encontrado
                        sea rb el siguiente elemento de cb
                        si ra y rb tienen la misma etiqueta
```

```
                    si ra y rb son de tipo cadena
                        si ra y rb tienen el mismo valor
                            hemos encontrado lo que buscábamos
                            añadimos el rasgo al conjunto de salida
                            eliminamos rb de cb
                        si no
                            devolver nulo
                        fin si
                    si no y ra y rb son de tipo conjunto de rasgos
                        sea cu el resultado de la unificación de los valores de ra y rb
                        si cu no es nulo
                            hemos encontrado lo que buscábamos
                            creamos nr con la misma etiqueta que ra y rb y con cu por valor
                            añadimos nr al conjunto de salida
                            eliminamos rb de cb
                        si no
                            devolver nulo
                        fin si
                    fin si
                fin si
                si no hemos encontrado el rasgo
                    añadimos ra al conjunto de salida
                fin si
            fin mientras
        fin mientras
        añadimos lo que quede en cb al conjunto de salida
        devolvemos el conjunto de salida
```

Por último en Unificador también tenemos un método que dado un conjunto de rasgos devuelve un String formateado correctamente con el contenido de dicho conjunto de rasgos en un formato fácilmente legible.

Como con el resto de clases, los detalles de implementación están perfectamente explicados en el código fuente.

Entorno

La clase Entorno es la que contiene el método por el que se inicia la ejecución del programa o método "main" y se encarga de conducir la ejecución del programa según los parámetros de entrada.

También es la clase que hace uso de todas las demás, coordinándolas para leer los ficheros y realizar las unificaciones pertinentes. Igualmente se encarga de esperar la introducción de comandos y realizarlos.

Como en el resto de clases, los detalles de la implementación se encuentran comentados en el código fuente.

Ejecuciones realizadas

Las ejecuciones las hemos realizado en diferentes ordenadores con diferentes características. A continuación, le indicamos las características de cada uno de los ordenadores, así como una tabla con los resultados de diez pruebas realizadas a cada uno de los tres test proporcionados. También incluimos la media de estas pruebas.

Características de los ordenadores:

Ordenador 1: Intel Core 2 Duo, 2,4 GHz, 2 GB
Ordenador 2: Intel Core i3, 3,2 GHz, 4GB
Ordenador 3: Intel Core 2 Duo, 2,26 GHz, 2 GB
Ordenador 4: Quad-Core Intel Xeon 4 núcleos, 2,93GHz, 8GB

		1	2	3	4	5	6	7	8	9	10	Media
1	T1	18	18	18	18	18	18	18	17	18	18	**17,9**
	T2	530	560	495	448	499	497	498	469	471	499	**496,6**
	T3	39327	39135	39051	39365	40479	39735	39070	39591	39193	39553	**39450**
2	T1	17	17	17	16	15	16	15	16	17	17	**16,3**
	T2	317	313	310	315	312	302	310	309	308	305	**310,1**
	T3	22369	22026	21992	21857	22094	21854	22011	22048	22256	22151	**22066**
3	T1	19	19	19	19	19	19	20	19	19	19	**19,1**
	T2	491	524	522	513	527	531	538	541	518	509	**521,4**
	T3	42127	47200	42828	43143	40443	40891	43360	40306	43398	41276	**42497**
4	T1	13	13	13	14	13	14	13	13	13	13	**13,2**
	T2	262	269	266	265	263	262	264	266	267	263	**264,7**
	T3	20528	20733	20495	20474	20456	20458	20493	20586	20270	20401	**20489**

Problemas Surgidos

Como ya hemos comentado anteriormente, tuvimos problemas con la memoria de java. Al principio guardábamos las estructuras ECR ya creadas después de ser parseadas, pero la memoria se consumía rápidamente. Optamos por guardar el String y crear aquellas ECR que realmente fuesen necesarias.

Cómo utilizar el Algoritmo

Para ejecutar el algoritmo se adjunta en la práctica un archivo java ejecutable "unificador.jar" que funciona ejecutado desde la consola.

A este ejecutable se le han de pasar entre 1 y 3 parámetros, de la siguiente forma y en cualquier orden:

Archivo ecr: -ecr <ruta_a_fichero_ecr>
Archivo unf: -unf <ruta_a_fichero_unf>
Archivo de salida: -o <ruta_a_fichero_de_salida>
Permitir la entrada de comandos: -c

De modo que se puede usar de dos formas diferentes:

1 - Pasándole únicamente el diccionario de ECRs, en cuyo caso aparecerá un prompt en la consola para que introduzcamos instrucciones del tipo "$Unifica(etiqeuta1 & etiqueta2)" que se ejecutarán y se mostrará el resultado por pantalla, aunque NO se escribirán en ningún fichero de salida. En este caso el parámetro "-c" se ignora.

2 - Pasándole el diccionario de ECRs, el archivo de instrucciones y el archivo de salida. En este caso, leerá el archivo unf, luego el ecr y escribirá los resultados en el archivo de salida, mostrándose en la consola lo que va sucediendo, pero NO el resultado de las unificaciones, que solamente aparece en el fichero de salida. En caso de que se haya pasado también el parámetro "-c", después de terminar todas las unificaciones apareceré el prompt a la espera de comandos de unificación, el resultado de estos comandos aparecerá en pantalla y NO se escribirán en el fichero de salida.

En entornos UNIX para ejecutar se utiliza el siguiente comando de consola:

java -jar <ruta_a_unificador.jar> <parámetros>