Capítulo 6
Recursividad

1. Introducción

Este capítulo es el segundo que introduce el concepto de *repetición* en los tratamientos de los datos después del capítulo «Iteración». Sin embargo, mientras que en los capítulos anteriores los cambios de estados del sistema se obtenían con instrucciones que indicaban explícitamente las modificaciones que se tenían que realizar en los datos, aquí se obtienen dando una *definición* de estos cambios. En particular, los tratamientos no usan la instrucción de asignación, salvo en contadas ocasiones, para dar un valor a la pseudovariable **`Resultado`** con el fin de precisar lo que calcula y devuelve una función.

La segunda sección introduce la recursividad en los tratamientos con cadenas de caracteres. Se muestra qué es una definición recursiva y se aplica este tipo de definición a la especificación de algoritmos con cadenas de caracteres. La siguiente sección da algunos ejemplos de especificaciones o de tratamientos recursivos con los números. Para terminar, la cuarta sección empieza el estudio de un problema con el que nos encontramos a menudo: la edición de un número. Se trata de transformar un número dado para obtener los caracteres que representan su valor en una base de numeración dada.

2. Introducción a la recursividad: las cadenas de caracteres

La recursividad se introduce en esta sección como un método general de especificación algorítmica de los algoritmos. Esta frase ya es por sí misma una frase que formula una propiedad recursiva: la especificación de un algoritmo es un algoritmo.

La primera parte presenta la recursividad con un ejemplo sencillo: la definición de una *palabra* del idioma. La segunda parte estudia ejemplos de especificaciones recursivas de algoritmos de tratamiento con las cadenas de caracteres. La tercera parte da la solución completa de algunos ejercicios didácticos y, finalmente, la última, propone resolver diversos ejercicios cuya solución no se ha desarrollado aquí.

2.1 Presentación de la recursividad

Consideremos una *palabra*. Una palabra es una cadena de caracteres de longitud finita. Para simplificar esta presentación, se restringe a las palabras, sean o no del idioma, escritas en minúsculas y sin caracteres acentuados: 'casa', 'pescado' o 'coche' son ejemplos de palabras que nos interesan. 'Colegio', 'Mansión' o 'elección' están excluidas porque contienen caracteres acentuados o mayúsculas.

¿Cómo dar una definición formal de una palabra?

Se trata de dar una definición que se aplica a una palabra cualquiera, sin acentos ni mayúsculas, como ya hemos mencionado. Se puede dar la siguiente definición:

Definición 1

Una palabra puede ser un carácter o un carácter seguido de una palabra.

Dicho de otra manera, una palabra, seleccionada entre las permitidas, puede ser una cadena de caracteres reducida a un único carácter o, si no, una cadena de caracteres formada por un primer carácter seguido de otra palabra.

Así, por ejemplo, la palabra 'casa' está formada por el carácter 'c' seguido de la palabra 'asa'. La palabra 'asa' está formada por 'a' seguido de la palabra 'sa' y así sucesivamente. Ahora es suficiente con decir qué es un carácter. Se puede definir un carácter mediante una enumeración.

Definición 2

Un carácter es un elemento del conjunto {'a', 'b',..., 'z'}.

En la definición de una palabra hay que destacar que utiliza palabra para definirla. Esta definición no es «circular»: dada una cadena de caracteres 'ca' de longitud finita, siempre es posible decidir, aplicando esta definición, si 'ca' es una palabra o no lo es.

Ejemplo

Aplicar esta definición a la cadena de caracteres 'lápiz' para determinar si es una palabra según la definición que hemos mencionado antes.

'lápiz' está formada por el carácter 'l' seguido de la cadena 'ápiz'. La cadena 'ápiz' está formada por 'á' seguida de la cadena 'piz', pero 'á' no es un carácter según la definición 2 dada por la enumeración que aparece más arriba. Por lo tanto 'lápiz' no es una palabra según la definición 1. Observe que, para este ejemplo, el resultado se ha obtenido cuando se ha encontrado el carácter «á», y no es necesario analizar los caracteres siguientes.

Ejemplo

¿Es una palabra la cadena de caracteres 'pan'?

Está formada por 'p', que es un carácter según la definición 2, seguido de la cadena 'an'. Esta está formada por el carácter 'a' seguido de la cadena 'n' y 'n' también es un carácter según la definición 2. Por lo tanto, 'pan' es una palabra.

Observe también que la cadena candidata debe tener al menos un carácter: entonces CADENA_VACÍA no es una palabra. Igualmente, NULO o CAR_VACÍO no son caracteres.

Considerando esta definición, ¿se puede escribir un predicado que reconoce si una cadena dada es una palabra? Puede hacerse «copiando» la definición anterior.

Primero se observa el primer carácter de la cadena y, si es el único, se puede concluir:

```
longitud(ca) = 1 => Resultado = es_un_carácter(primero(ca))
```

La función **primero** devuelve el primer carácter de la cadena que recibe como parámetro. La definiremos más tarde. El predicado **es_un_carácter** devuelve VERDADERO si y solo si el parámetro es un carácter autorizado. Así, cuando el primer carácter de la cadena, que también es el único, es un carácter según la definición 2, **es_un_carácter** devuelve VERDADERO y este es el valor que debe tomar **Resultado**.

Cuando la cadena está compuesta de muchos caracteres, se verifica que el primer carácter está autorizado y luego que el resto de la cadena es una palabra:

```
longitud(ca) > 1 => Resultado =
    (
        es_un_carácter(primero(ca))
      y entonces
        es_una_palabra(fin(ca))
    )
```

La función **fin** devuelve la cadena, que ha recibido como parámetro, sin el primer carácter. También definiremos esta función con precisión más adelante.

La definición 1 dice que un carácter autorizado es una palabra. No podemos escribir directamente:

```
# INICIO DE SOLUCIÓN INCORRECTA !!!
longitud(ca) > 1 => Resultado =
    (
        es_una_palabra(primero(ca))
      y entonces
        es_una_palabra(fin(ca))
    )
# FIN DE SOLUCIÓN INCORRECTA
```

porque **es_una_palabra** espera una cadena de caracteres como parámetro, pero **primero** devuelve un **CARÁCTER**. El uso de **es_una_palabra** con el resultado de **primero** como parámetro es incorrecto porque no respeta las condiciones de uso de **es_una_palabra** precisadas en sus especificaciones.

Finalmente, se obtiene el siguiente algoritmo:

Algoritmo 1: Reconocer una palabra: ***es_una_palabra*** *- Versión 1.0*

```
Algoritmo es_una_palabra
    # ¿'ca' es una palabra?

Entrada
    ca: CADENA

Resultado: BOOLEANO

precondición
    ca ≠ NULO
    longitud(ca) ≥ 1

poscondición
    longitud(ca) = 1 => Resultado = es_un_carácter(primero(ca))
    longitud(ca) > 1 => Resultado =
    (
        es_un_carácter(primero(ca))
      y entonces
        es_una_palabra (fin(ca))
    )

fin es_una_palabra
```

La poscondición también se puede escribir:

```
poscondición
    (∀k∈ℕ)(1 ≤ k ≤ longitud (ca) => es_un_carácter(ítem(ca, k))
```

Esta nueva versión de la poscondición supone que los caracteres de la cadena están numerados a partir de 1. Si su numeración es cualquiera, se usará **índice_min**(ca) e **índice_max**(ca), como aquí:

```
poscondición
    (∀k ∈ ℤ)(índice_min(ca) ≤ k ≤ índice_max(ca) =>
                                        es_un_carácter(ítem(ca, k))
```

Volvamos al algoritmo.

No habrá asignación en la realización, es decir, ninguna orden para ejecutar, salvo quizás, de una manera marginal, a causa de las convenciones de notación, la asignación de un valor a la pseudovariable **Resultado** para indicar el resultado calculado y devuelto por la función.

La realización del algoritmo es sencilla: solo copia la especificación que únicamente traduce de una manera algorítmica las definiciones dadas más arriba. Esta realización es la siguiente:

```
realización
    si
        longitud(ca) = 1
    entonces
        Resultado ← es_un_carácter(primero(ca))
    si no
        Resultado ←
            (
                es_un_carácter(primero(ca))
              y entonces
                es_una_palabra(fin(ca))
            )
    fin si
```

Este algoritmo no dice cómo calcular el resultado, incluso dentro de su realización. Solo da una definición. Para eso, utiliza sus propios servicios, en la misma cadena de caracteres, pero a partir de un carácter de número superior. Una definición que llama a un concepto para definirlo, como la definición 1 define *palabra* usando *palabra*, es una *definición recursiva*. Asimismo, un algoritmo que llama a sus propios servicios es un algoritmo *recursivo*.

Definición 3

Se dice de un algoritmo que es recursivo simple cuando se llama a sí mismo para realizar sus cálculos.

Hay formas más complejas de recursividad que trataremos más adelante.

Observe cómo se construye la poscondición de **es_una_palabra**. Empieza por dar el resultado cuando la cadena de caracteres está formada por un carácter único. La cadena es una palabra, según la definición 1, si su único carácter es un carácter definido en la enumeración de la definición 2.

Este caso de cadena de longitud 1 es el caso particular que permite devolver inmediatamente en resultado. Cuando la cadena está hecha de varios caracteres, el resultado toma el valor `VERDADERO` si el primer carácter es un carácter aceptable, según la definición 2, y si la cadena, privada de este primer carácter, también es una palabra. Es exactamente la definición 1. En cada ensayo de **`es_una_palabra`**, se amputa el primer carácter de la cadena.

La longitud de las cadenas analizadas decrece de manera estricta en una unidad. Por lo tanto, como una cadena es, por definición, de longitud finita, un número finito de operaciones lleva a una cadena de longitud 1, y entonces termina el algoritmo. Incluso es posible dar una cota superior exacta de la cantidad de operaciones al final de las que el algoritmo produce su resultado: es **`longitud`**`(ca)`. Así, el algoritmo termina con una cadena de longitud 1 y, en todo momento, es decir, en cada estado del sistema de software, la cantidad de estados que quedan por recorrer es igual, como máximo, a la longitud de la cadena residual.

Resumiendo: la longitud de la cadena es un número entero finito, que decrece estrictamente en una unidad con cada llamada a la función y que se reduce en una longitud igual a 1. Este análisis prueba que el algoritmo termina en un número finito de etapas.

De esta manera, el algoritmo converge hacia el resultado, pero aquí hemos aprendido algo fundamental: sabemos medir el número máximo de estados que hay que visitar antes de obtener el resultado. Como máximo, es la longitud de la cadena residual en el estado considerado.

Lo hemos hecho todo para que esta solución sea correcta y podemos tener una confianza razonable en que dará el resultado esperado; sin embargo, podemos estudiar su funcionamiento en un ejemplo para asegurarnos. Este ejemplo será el caso de la cadena `'lápiz'` estudiada en la introducción de este capítulo. El cálculo da **`Resultado`** `= FALSO` cuando se encuentra `'á'`.

Para simplificar las notaciones, **`es_un_carácter`**`('k')` se anotará como `c('k');` **`es_una_palabra`**`(ca)`, donde `ca` es una cadena de caracteres, se anotará como `m(ca)`; la longitud de `ca` se anotará como `L(ca)`.