

Autómatas y Lenguajes Formales 2019-II

Facultad de Ciencias UNAM*

Nota de Clase X: Listas abstractas y concretas

Favio E. Miranda Perea A. Liliana Reyes Cabello
Lourdes González Huesca

-1 de diciembre de 2020

1. Abstracción lingüística

Si uno recuerda un lenguaje de programación con el que se está familiarizado, se puede observar que, aunque superficialmente diferentes en sus *palabras reservadas* y *separadores*, son usualmente bastante similares en un nivel más profundo. Cambiando el enfoque de sintaxis concreta a sintaxis abstracta, podemos reducir la desconcertante variedad de construcciones del lenguaje a algunas estructuras esenciales. El verbo “abstraer” significa

considerar un concepto sin pensar en un ejemplo específico.

Abstrayendo de los caracteres concretos que representan una construcción del lenguaje se hace la abstracción lingüística. Esto es la transformación del lenguaje que remplaza los caracteres terminales de un lenguaje concreto con otros caracteres tomados de una alfabeto abstracto. Los caracteres abstractos deben ser más simples y adecuados para representar construcciones similares de diferentes lenguajes artificiales¹.

Mediante este enfoque, las estructuras de sintaxis abstractas de lenguajes artificiales existentes son fácilmente descritas como composición de algunos paradigmas básicos, mediante operaciones estándares del lenguaje: unión, iteración y sustitución. Empezando de un lenguaje abstracto, un lenguaje concreto o existente es obtenido por la transformación de inversa, metafóricamente llamada recubriendo con azúcar sintáctica.

*Material elaborado en el marco del proyecto PAPIME PE102117

¹La idea de la abstracción del lenguaje se inspira en la investigación en lingüística con el objetivo de descubrir las similitudes subyacentes de los lenguajes humanos, sin tener en cuenta las diferencias morfológicas y sintácticas.

Fabricando un lenguaje en su sintaxis abstracta y sintaxis concreta se compensa en varias maneras. Cuando se estudian diferentes lenguajes se obtiene mucho ahorro conceptual. Cuando se diseñan compiladores, la abstracción ayuda para la portabilidad entre diferentes lenguajes, si las funciones del compilador están diseñadas para procesar lo abstracto, en vez de lo concreto, el lenguaje construye. Por lo tanto partes de, digamos, el compilador de C puede ser reusado para lenguajes similares como FORTRAN o Pascal.

2. Listas abstractas y concretas

Una *lista abstracta* contienen un número no acotado de elementos e del mismo tipo. Está definido por la expresión regular e^+ o e^* , si pueden faltar elementos.

Un elemento por el momento puede ser visto como un carácter terminal, pero en refinamientos posteriores, el elemento se puede convertir en una cadena de otro lenguaje formal: pensemos, por ejemplo, en una lista de números.

Lista con separadores y símbolos que abren/cierran.

En muchos casos concretos, los elementos adyacentes deben ser separados por cadenas llamadas *separadores*, s en sintaxis abstracta. Por lo tanto en una lista de números, un separador debe delimitar el final de un número y el principio del siguiente.

Una *lista con separadores* es definida por la expresión regular $e(se)^*$, diciendo que el primer elemento puede ser seguido por cero o más pares se . La definición equivalente $(es)^*e$ difiere dando evidencia al último elemento.

En muchos casos concretos hay otro requisito, destinado a la legibilidad o el procesamiento por computadora: hacer el inicio y final de una lista fácilmente reconocible agregando algunos signos especiales a sufijos y prefijos: en abstracto, el carácter inicial o la marca de inicio i , y el carácter final o la marca final f .

Listas con separadores y marcas inicial/final son definidas como

$$ie(se)^*f$$

Ejemplo: Algunas listas concretas

Las listas están en todos lados en los lenguajes, como es mostrado por ejemplos típicos.

Bloque de instrucciones: $\begin{array}{l} \text{begin } instr_1; instr_2; \dots instr_n \text{ end} \end{array}$

donde $instr$ posiblemente está ahí para asignaciones, go to, sentencia **if**, sentencia **write**, etc. Los términos abstractos y concretos correspondientes son:

Parámetros de procedimientos: como en

$$\underbrace{\text{procedure } WRITE}_{i}(\underbrace{\text{par}_1}_{e}, \underbrace{\text{,}}_{s}, \underbrace{\text{par}_2, \dots, \text{par}_n}_{f})$$

alfabeto abstracto	alfabeto concreto
i	$begin$
e	$instr$
s	$;$
f	end

Una lista vacía de parámetros debe ser legal, como por ejemplo *procedure WRITE()*, la expresión regular se convierte en $i[e(se^*)]f$.

Definición de arreglo: $\underbrace{\text{array } MATRIX'}_i[' \underbrace{\text{int}_1}_e \underbrace{,}_{s} \underbrace{\text{int}_2, \dots, \text{int}_n}_{f} \underbrace{']}_f$

donde cada int es un intervalo como 10...50.

2.1. Sustitución

Los ejemplos de arriba muestran la transformación de símbolos concretos a abstractos. Los diseñadores de lenguajes encuentran útil trabajar con refinamientos paso a paso, como es hecho en cualquier rama de ingeniería, cuando un sistema complejo es dividido en sus componentes, atómico o de otros tipos. Para tal fin, se presentan una nueva operación del lenguaje de sustitución, que remplaza un carácter terminal del lenguaje denominado como *fuente*, con una sentencia de otro lenguaje llamada *objetivo*. Como siempre Σ es el alfabeto fuente y $L \subseteq \Sigma^*$ el lenguaje fuente. Consideremos una sentencia de L conteniendo una o más ocurrencias de un carácter fuente b :

$$x = a_1 a_2 \dots a_n \quad \text{donde para alguna } i, a_i = b$$

Sea Δ otro alfabeto, llamado objetivo, y $L_b \subseteq \Delta^*$ sea el *lenguaje imagen* de b . La *sustitución* del lenguaje L_b para b en una cadena x produce un conjunto de cadenas, que es, una lenguaje del alfabeto $(\Sigma \setminus \{b\}) \cup \Delta$, definido como

$$\{y \mid y = y_1 y_2 \dots y_n \wedge (\text{if } a_i \neq b \text{ then } y_i = a_i \text{ else } y_i \in L_b)\}$$

Notemos que todos los caracteres que no sean b no cambian. La sustitución puede ser definida en todo el lenguaje fuente, aplicando la operación a cada sentencia objetivo.

Ejemplo: El ejemplo anterior se retomará

Regresando al caso de lista de parámetros, la sintaxis abstracta es

$$ie(se)^*f$$

y las sustituciones a ser aplicadas se muestran a continuación:

caracter abstracto	cadena
i	$L_i = \text{procedure}(\text{procedure identifier})($
e	$L_e = (\text{parameteer identifier})$
s	$L_s = ,$
f	$L_f =)$

Por ejemplo, la marca de inicio de i es remplazada con una cadena del lenguaje L_i , donde **procedure identifier** tiene que estar de acuerdo con las reglas del lenguaje técnico.

Claramente los lenguajes objetivos de la sustitución dependen en la azucar sintáctica del lenguaje concreto al que está destinado.

Cabe notar que las cuatro sustituciones son independientes y pueden ser aplicadas en cualquier orden.

Ejemplo: Identificadores con guiones bajo.

En ciertos lenguajes de programación, los nombres de identificadores nmemotécicos largos pueden ser construidos añadiendo cadenas alfanuméricas separadas por un guión bajo: de ahí que `LOOP3_OF_35` es un identificador válido. Más precisamente la primera cadena debe iniciar con una letra, los otros deben contener letras y dígitos, y los guiones adyacentes están prohibidos, así como guiones finales.

A primera vista el lenguaje es una lista no vacía de cadenas s , separadas por un guión:

$$s(-s)^*$$

Sin embargo, la primera cadea debe ser diferente de las otras y debe ser tomada para ser la marca de inicio de una posible lista vacía:

$$i(-s)^*$$

Sustituyendo a i el lenguaje $(A \dots Z)(A \dots Z|0 \dots 9)^*$, y a s el lenguaje $(A \dots Z)(A \dots Z|0 \dots 9)^+$, la expresión regular final es obtenida.

Este es un ejemplo demasiado simple de diseño de sintaxis por abstracción y refinamiento paso a paso, un método que será utilizado ahora y después después de la introducción de gramáticas.

2.2. Listas de precedencia o jerárquicas

Una construcción recurrente es una lista, tal que cada elemento que está en turno es una lista de una de tipo diferente. La primera línea está en el nivel 1, la segunda al nivel 2, y así sucesivamente. La estructura abstracta presente, llamada lista jerárquica, está resstringida a listas con un número acotado de niveles. El caso en el que los niveles no

están acotados es estudiado después usando gramáticas, bajo el nombre de estructuras anidadas.

Una lista jerárquica es también llamada *lista con precedencias*, porque una lista a nivel k limita sus elementos más que a una lista a nivel $k - 1$; en otras palabras los elementos en un más alto nivel deben ser juntados en una lista, y cada uno se convierte en un elemento al siguiente nivel más bajo.

Cada nivel debe tener una marca que abra/cierre y un separador; tales delimitadores son usualmente distintos nivel por nivel, para evitar confusiones.

La estructura de la lista jerárquica de niveles $k \geq 2$ es

$$\begin{aligned} list_1 &= i_1 list_2 (s_1 list_2)^* f_1 \\ list_2 &= i_2 list_3 (s_2 list_3)^* f_2 \\ &\dots \\ list_k &= i_k e_k (s_k e_k)^* f_k \end{aligned}$$

Nótese que sólo el último nivel puede contener elementos atómicos. Pero una variación común permite que a cualquier nivel k los elementos atómicos e_k ocurran al lado de otro con listas de nivel $k + 1$. A continuación se presentan algunos ejemplos concretos.

Ejemplo: Dos listas jerárquicas

Bloque de instrucciones `print`, $WRITE(var_1, var_2, \dots, var_n)$, es decir, una lista. Hay dos niveles:

Nivel 1: Lista de instrucciones *instr*, abierta por *begin*, separados por un punto y coma y cerrado por *end*.

Nivel 2: Lista de variables *var* separados por una coma, con $i_2 = WRITE($ y $f_2 =)$.

Las expresiones aritméticas no usan paréntesis: los niveles de precedencia determinan de los operadores determinan cuantos niveles hay en la lista. Por ejemplo, los operadores \div , \times y $+$, $-$ están etiquetados en dos niveles y la cadena

$$3 + \underbrace{5 \times 7 \times 4}_{term_1} - \underbrace{8 \times 2 \div 5}_{term_2} + 8 + 3$$

es una lista de dos niveles, sin marcas de apertura ni de cierre. En el nivel uno encontramos una lista de términos ($e_1 = \text{término}$) separados por los signos $+$ y $-$, es decir, por operadores de menor precedencia. En el nivel dos vemos una lista de números, separados por signos de mayor precedencia.

Se puede ir más allá e introducir un tercer nivel con el signo de exponentiación “**” como separador.

Por supuesto, las estructuras jerárquicas también son omnipresentes en los lenguajes naturales. Piense en una lista de sustantivos

padre, madre, hijo, e hija

Aquí podemos observar una diferencia con respecto al paradigma abstracto: el penúltimo elemento tiene un separador distinto, posiblemente para advertir al oyente de que ya vamos a acabar de hablar. Además, los elementos de la lista pueden enriquecerse con calificadores de segundo nivel, como la lista de adjetivos.

En todo tipo de documentos y medios escritos, las listas jerárquicas son extremadamente comunes. Por ejemplo, un libro es una lista de capítulos, separados por páginas en blanco, entre la portada y la contraportada. Un capítulo es una lista de secciones; una sección, una lista de párrafos, y así sucesivamente.