Práctica 1

Introducción

El lenguaje de programación escogido para las prácticas es Mathematica [Wolfram,91]. En las implementaciones que se van a llevar a cabo, hay que manejar objetos abstractos como son los autómatas y las gramáticas, sobre los que se realizarán una serie de transformaciones. Tanto unos como otras pueden ser concebidos como estructuras más complejas realizadas en base a otras más sencillas. El interés de las prácticas no estriba en conseguir implementaciones eficientes, sino en utilizar un lenguaje que facilite la construcción de estos objetos abstractos y el manejo de los mismos. Teniendo en cuenta todos estos aspectos, veamos a continuación algunas de las características de Mathematica que lo hacen adecuado para su uso en dichas prácticas.

Mathematica es un paquete de desarrollo para aplicaciones de tipo general en las que los aspectos de desarrollo matemático, algebraico, numérico, simbólico y gráfico juegan un papel preponderante. La aplicaciones de Mathematica engloban prácticamente todas las áreas de investigación y desarrollo tanto en investigaciones científicas, de ingeniería, económicas, arquitectura, etc. Dentro de estas mismas áreas se pueden utilizar tanto como herramienta de trabajo como herramienta docente para aquellas materias que conlleven un alta carga de desarrollo matemático.

Mathematica como lenguaje de programación se diferencia de los lenguajes de propósito general, por una parte por su habilidad en el tratamiento de expresiones matemáticas, números, expresiones simbólicas, etc., y por otra en que es un lenguaje interpretado. Como consecuencia de ser interpretado, un cálculo tarda más tiempo en ejecutarse que en un lenguaje compilado, sin embargo, el escribir un programa en Mathematica requiere una fracción del tiempo necesario para escribir el mismo programa en otro lenguaje, y lo que es más importante, en opinión del equipo docente, permite concentrar los esfuerzos en los detalles conceptuales y no en los de implementación.

El sistema Mathematica es interactivo, lo cual significa que no hay que compilar programas. En lugar de eso, los cálculos se hacen típicamente ejecutando las expresiones necesarias, y por tanto, los resultados intermedios de estas evaluaciones pueden verse inmediatamente. En esta primera práctica se pretende introducir algunos conceptos elementales del lenguaje de programación incluído en Mathematica, así como el desarrollo de programas relacionados con palabras y lenguajes empleando esta herramienta.

Antes de empezar

La ventana de Mathematica es totalmente típica. Sin embargo, tiene algunas peculiaridades. Las más importantes son:

- Mathematica es sensible a mayúsculas, por lo que los identificadores son distintos si se escriben con mayúsculas que si lo hacen con minúsculas.
- Para evaluar/ejecutar una instrucción hay que teclear **shift+intro**.
- Las expresiones se van numerando de manera automática en función del momento de evaluación y se disponen en **celdas**. Cada instrucción lleva asociada una celda de

entrada (In[..]) y otra de salida (Out[..]). Su disposición se puede ver en lineas verticales a la derecha.

- Mientras dura la ejecución de una intrucción la linea de la celda de entrada permanece con trazado doble. Si se entra en un bucle indefinido se puede interrumpir la ejecución tecleando **Alt+.** y, si esto no funciona, con **Quit Kernel** del menú **Evaluation**. La primera de las opciones solo interrumpe la ejecución en curso, mientras la segunda anula todas evaluaciones las de la sesión actual.

- En todo momento se puede pedir ayuda interactiva sobre la sintaxis de un comando escribiendo el signo **?** seguido del nombre del comando (o parte de él, terminando con ***).

- Mathematica colorea el código de acuerdo a la sintaxis esperada y sangra el texto de manera automática cuando se realizan los programas. Si el texto está mal sangrado suele deberse a sintaxis incorrecta.

Listas en Mathematica

Una de las estructuras de datos más importantes en Mathematica (y, sin duda la que nosotros emplearemos más) es la lista. Se pueden emplear listas tanto para las palabras de los lenguajes con los que trabajemos, como para las transiciones de los autómatas que reconozcan dichos lenguajes.

Las listas se representan entre llaves y sus elementos están separados por comas, por ejemplo: list1 = {a, b, {c, d}} asigna a la variable **list1** la lista con primer elemento **a** , segundo elemento **b** y tercero la lista {c, d}. El elemento i-ésimo de una lista se referencia como nombre[[i]]. Así, list1[[3]] es {c, d}. Obsérvese que los elementos de la lista no tienen que ser homogéneos.

Operaciones con listas

Para ahorrar tiempo comenzamos contruyendo listas de manera automática. Con la instrucción `l=Table[Random[Integer, {1, 9}], {i, 20}]` (**shift+intro**) asignamos a la variable **l** una lista de 20 números enteros tomados aleatoriamente en el rango de 1 a 9 (naturalmente se puede variar tanto la cantidad como el rango).

Constrúyanse dos listas **l1** e **l2**, y pruébense los siguientes comandos:

Importante: los comandos que se describen a continuación, salvo los que se indican, no actualizan las listas. Para poder disponer del resultado hay que asignarlo a una variable.

- `Length[l1]`: Devuelve la longitud de la lista.

- `Join [l1, l2]`: Concatena dos listas.

- `Union[l1, l2]`: Devuelve una lista con los elementos que se encuentran en l1 o l2 y los ordena.

- `Intersection[l1, l2]`: Devuelve una lista con los elementos comunes a l1 y l2

- `Complement[l1, l2]`: Devuelve una lista con los elementos de l1 que no estan en l2.
- `Sort[l1]`: Devuelve l1 ordenada de menor a mayor (no actualiza l1).
- `Reverse[l1]`: Devuelve el reverso de l1.
- `RotateRight[l1]`: Devuelve l1 con los elementos desplazados un lugar a la derecha (el último pasa a ser el primero).
- `RotateLeft[l1]`: Idéntico al anterior pero desplazando hacia la izquierda
- `First[l1]`: Devuelve el primer elemento de la lista.
- `Rest[l1]`: Lista l1 sin el primer elemento.
- `Drop[l1, n]`: Devuelve la lista sin los primeros n elementos.
- `Take[l1, n]`: Devuelve los primeros n elementos de la lista.
- `Append[l1, x]`: Añade el elemento x al final.
- `Prepend[l1, x]`: Añade el elemento x al comienzo.
- `AppendTo[l1, x]`, `PrependTo[l1, x]`: Idénticas a las anteriores pero actualizan la lista.
- `Position[l1,x]`: Devuelve una lista con las posiciones de x en l1.
- `MemberQ[l1,x]`: Devuelve True si x pertenece a l1 y False si no.

La función Cases

Por su importancia, destacamos la siguiente función:

`Cases[lista, patrón]`: Devuelve una lista con los elementos de lista que concuerdan con patrón. El patrón puede contener el símbolo _ (subrayado), que se sustituye por cualquier símbolo.

Ejemplo `lista={{a,a},{b,a},{b,b},{a,b}}`
`Cases[lista,{a,_}]` devuelve {{a,a},{a,b}}.
`Cases[lista,{c,_}]` devuelve {} (lista vacía).

Operadores lógicos y relacionales

Son operadores que dan como resultado True o False.
Son los siguientes:

- Negación `!`
- Conjunción `&&`
- Disyunción `||`

- Igualdad `==`
- No igualdad `=!=`
- `>, <, >=, <=`.

Programación

Mathematica lleva incorporado un lenguaje de programación propio que permite incorporar funciones para realizar tareas específicas al mismo nivel que las funciones predefinidas.

Al ser un entorno que interpreta las expresiones, el modo de trabajo puede ser totalmente interactivo; Así, si ejecutamos la expresión `For[i = 1, i < 10, i++, Print[i]]` se escriben en pantalla los dígitos del 1 al 9 (sirva como ejemplo la instrucción for para indicar como la sintaxis en Mathematica es muy similar a la de otros lenguajes de programación habituales).

Módulos

El concepto de módulo es totalmente parecido al de método o función en otros lenguajes de programación. Su esquema genérico es:

`nombre[parámetros] : = Module [{`variables locales separadas por comas`},`
Acciones (separadas por `;`);
`Return[nvar]` (si el módulo devuelve un valor)
`]`

El módulo, las variables y la ejecución del módulo con las variables conviene que estén en celdas diferentes.

Ejemplo

Módulo que toma como entrada un número positivo n y devuelve la suma de los n primeros números enteros.

```
suma[n_Integer]:=Module[{i,suma1},
   suma1=0;
   For[i = 1, i <= n, i++,
      suma1 = suma1 + i;
   ];
   Return[suma1];
]
```

Para ejecutarlo se escribiría por ejemplo `suma[3];` (que dará como resultado 6).

Estructuras condicionales y de repetición

- **Condicional**: `If[condición, sentencias-verdad, sentencias-falso]`
- **Iteración fija**: `For[comienzo, test, incremento, sentencias]`; Se evalúa *comienzo* y se ejecutan *sentencias* e *incremento* hasta que *test* falla.
- **Iteración** `While[condición, sentencias]`; Se ejecuta *sentencias* mientras *condición* es cierta.

Representación de palabras y lenguajes finitos

En lo que sigue, una palabra se representa como una una lista de símbolos sobre un determinado alfabeto. Así la palabra *x=abbaca*, se representará como `{a,b,b,a,c,a}` y la palabra vacía se representa como la lista de longitud 0, es decir, `{}`. Un lenguaje finito es un conjunto finito de palabras. Por tanto, un lenguaje se representa como una lista cuyos elementos (palabras) son listas. Por ejemplo, el lenguaje `L={abba, bb}`, se representará como `{{a,b,b,a},{b,b}}`, el lenguaje vacío se representa como `{}` y el lenguaje que solo tiene la palabra vacía se representará como `{{}}`.

Ejercicios

Ejercicio 1

Escriba un módulo Mathematica que con entrada una palabra x y un símbolo a, calcular $|x|_a$ (número de ocurrencias de a en x).

Ejercicio 2

Escriba un módulo Mathematica que con entrada una palabra x y un entero positivo n, obtenga x^n (concatenación de la palabra x consigo misma n veces).

Ejercicio 3

Escriba un módulo Mathematica que devuelva el conjunto de prefijos de una palabra x.

Ejercicio 4

Escriba un módulo Mathematica que devuelva el conjunto de sufijos de una palabra x.

Ejercicio 5

Escriba un módulo Mathematica que devuelva el conjunto de segmentos de una palabra x.

Ejercicio 6

Escriba un módulo Mathematica que devuelva el producto de dos lenguajes finitos dados.

Ejemplo: Dados $L_1 = \{a, bb, aba\}$ y $L_2 = \{\lambda, ba, bba\}$, el módulo deberá devolver $\{a, bb, aba, abba, bbba, ababa, bbbba, ababba\}$.

Téngase en cuenta que, en estas prácticas, la representación de las palabras es mediante listas de símbolos y la representación de lenguajes considera listas de palabras, por lo tanto, listas de listas.

Ejercicio 7

Escriba un módulo Mathematica que devuelva la unión de dos lenguajes finitos dados.

Ejemplo: Dados $L_1 = \{a, bb, aba\}$ y $L_2 = \{\lambda, bb, bba\}$, el módulo deberá devolver $\{\lambda, a, bb, aba, bba\}$.

Ejercicio 8

Escriba un módulo Mathematica que, con entrada un lenguaje finito L, y un entero $n > 0$ calcular L^n.

Ejemplo: Dados $L_1 = \{a, bb, aba\}$ y $n = 2$, el módulo deberá devolver $\{aa, abb, aaba, bba, bbbb, bbaba, abaa, ababb, abaaba\}$.

Ejercicio 9

Diseñe e implemente un módulo *Mathematica* que, dada una palabra x sobre el alfabeto $\{a, b\}$ como entrada, devuelva $True$ si x contiene un número par de símbolos a y al menos dos símbolos b. En caso contrario el módulo devolverá $False$.

Ejemplo: Dada $x = \{a, b, a, b, b, a, a, a, b, a, b, a, b, b, a, a, b, a, b, a, a, a, a, b, a, a, b, a\}$, el módulo deberá devolver $False$

Ejercicio 10

Diseñe e implemente un módulo *Mathematica* que, dada una palabra x sobre el alfabeto $\{a, b, c\}$ como entrada, devuelva $True$ si x contiene un número par de símbolos b después del último simbolo c. En caso contrario el módulo devolverá $False$.

Ejemplo: Dada $x_1 = \{a, b, a, c, b, b, a, a, a, b, c\}$, el módulo deberá devolver $True$. Dada $x_2 = \{a, b, a, c, b, b, a, a, a, b, c, a, a, b, a, a, a\}$, el módulo deberá devolver $False$. Dada $x_3 = \{a, b, a, b, b, a, a, a, b, a, a, a, a\}$ el módulo deberá devolver $False$ (x_3 no contiene el símbolo c).

Ejercicio 11

Se dice que una palabra s es subpalabra de x si s denota una secuencia de símbolos que aparecen en x en ese orden aunque no necesariamente consecutivos.

Diseñe e implemente un módulo *Mathematica* que, dadas dos palabras x y s sobre el alfabeto $\{a, b\}$, devuelva $True$ si x contiene la subpalabra s. En caso contrario el módulo devolverá $False$.

Ejemplo: Dada la palabra $x = \{a, b, a, a, b, a, a, b, a\}$, algunas subpalabras de x son $\{a, b, a\}$, $\{a, a, a\}$, $\{b, b, a\}$ o $\{b, b, b\}$. Sin embargo, $\{b, b, b, a, a\}$ no es subpalabra de x.

Ejercicio 12

Escriba un módulo Mathematica que, dadas dos palabras x_n y x_m de longitudes n y m, con $n \leq m$, devuelva $False$ si x_n no es un segmento de x_m, o bien, caso que sí lo sea, la posición del primer símbolo de x_n en x_m.

Ejemplo: Dadas $x_n = \{b, a, a, b\}$ y $x_m = \{b, a, b, a, b, b, b, a, b, a, a, b, b, b, b, a, b, a, b, b, a, a, b, a, a\}$, el módulo deberá devolver 9.

Ejercicio 13

Escriba un módulo Mathematica que, dados un conjunto de palabras M y una palabra x, devuelva el conjunto de posiciones de x donde se puede encontrar una palabra de M como segmento así como el segmento encontrado.

Ejemplo: Dados:

$$M = \{\{b, b\}, \{a, b, b, b\}, \{b, b, a, b\}, \{a, a, a, a\}\}$$

$$x = \{b, a, b, a, a, b, b, a, b, b, b, a, b, b, a, b, a, a, a, a, a, b, b, a, a, b, b, a, b, a\}$$

el programa debería devolver:

$$\{\{6,\{b,b\}\},\{6,\{b,b,a,b\}\},\{8,\{a,b,b,b\}\},\{9,\{b,b\}\},\{10,\{b,b\}\},\{10,bbab\},$$
$$\{17,\{a,a,a,a\}\},\{18,\{a,a,a,a\}\},\{22,\{b,b\}\},\{26,\{b,b\}\},\{26,\{b,b,a,b\}\}\}$$