uc3m | Universidad Carlos III de Madrid

PRÁCTICA JFLAP
SESIÓN 3
Curso 2019-2020

1. OBJETIVO

El objetivo principal de esta práctica es conocer la aplicación de la Teoría de Autómatas y Lenguajes Formales para el desarrollo de compiladores. Para ello se requiere el diseño de la gramática correspondiente al analizador sintáctico de un compilador para un determinado lenguaje de programación. En este caso proponemos un pequeño prototipo del lenguaje ***awk***. Se trata de un lenguaje/herramienta orientado al procesamiento de ficheros de texto que se encuentra en cualquier sistema operativo Unix.
Dado que ahora mismo no disponemos de los medios para construir un compilador operativo, nos limitaremos a utilizar la herramienta **JFLAP** para comprobar la corrección sintáctica de programas elementales escritos en dicho lenguaje. Los requisitos del lenguaje se especifican a continuación.

2. ESTRUCTURA DE UN PROGRAMA

La estructura básica del lenguaje awk consiste en una secuencia de pares patrón-acción. Cada patrón especifica algo que debe encontrarse en el fichero de texto que se procesa para desencadenar la acción correspondiente. Un caso de patrón es la etiqueta BEGIN. Cualquier acción asociada a dicha etiqueta, será aplicada al inicio de procesar un fichero. La etiqueta END marca una acción que será aplicada al finalizar el proceso.

Hay patrones mucho más complejos de lo que nos permite abordar esta práctica, por lo que como tercera opción consideraremos el patrón por omisión (lambda) que se asocia a las acciones que deberán ejecutarse para cada línea del fichero procesado.

Una forma de uso muy habitual consistirá en llamar al programa awk con la secuencia de instrucciones entre comillas simples y el fichero que se desea procesar.

Ejemplo de uso:

```
awk 'BEGIN{j=0;} {j=j+1;} END{print j;}'  lista_telefonos.txt
```

`awk`	programa que procesa ficheros con la secuencia de instrucciones entre comillas simples
`BEGIN {j=0;}`	Secuencia inicial que inicializa la variable j a 0
`{j=j+1;}`	Secuencia de instrucciones que se aplicará por cada línea del fichero de datos leído. En este caso, incrementa la variable j en una unidad por cada línea
`END {print j;}`	Imprime el valor de j una vez finalizada la lectura del fichero.
`lista_telefonos.txt`	Fichero que contiene un teléfono por línea.

El proceso que realiza el programa es contar el número de líneas del fichero de teléfonos.

Otro ejemplo:

```
awk  '{print $1; print $3; }'  lista_telefonos.txt
```

Suponiendo que nuestro fichero de teléfonos contiene en cada línea cuatro campos con:

Nº de teléfono, Nombre, Apellido1, Apellido2

El procesamiento dará como resultado una salida que contendrá únicamente el número de teléfono y el primer apellido de cada línea (campos que se indexan con los parámetros $1 y $3 respectivamente)

El lenguaje nos permite emplear diversos elementos:

- Variables (j, k, m, n)
- Parámetros ($0, $1, $2, …)
- Valores enteros (0, 1, …)
- Operadores de asignación (=), comparación (==, !=, <, >) y aritméticos (+, -, *, /, %).
- Operación de impresión (print)
- Estructura de control condicional (if)

Otro ejemplo:

```
awk  'BEGIN{j=0;} {j=j+1; if(j%2==0) print $0;}'  lista_telefonos.txt
```

Sirve para imprimir las líneas pares del listado telefónico ($0 representa una línea entera).

Especificaciones más detalladas:

Enumeramos las especificaciones del lenguaje que nos van a ser necesarias:

1. Un *Programa* se compone de uno o varios pares *Patrón Acción*, donde la *Acción* es un *Bloque* entre llaves.

2. Un *Patrón* puede ser una etiqueta BEGIN o END, o también puede ser *lambda*.

3. Un *Bloque* contendrá una o más *Sentencias*.

4. Una *Sentencia* puede pertenecer a tres tipos: *Asignación*, *Impresión* o *Condicional*. Estarán siempre terminadas por un ; (punto y coma).

5. *La Asignación* sirve para asignar el valor de una *Expresión* a una *Variable* mediante el operador = (igual).

6. La *Impresión* permite imprimir el valor de una *Expresión* con la palabra reservada *print.*

7. Una *Condicional* comienza por la palabra *if* seguida de una *Condición* y de una *Sentencia*.[1]

8. Una *Expresión* puede derivar en un *Término.* Una *Expresión* también puede ser la combinación de dos *Expresiones* con un *Operador* intercalado y englobada opcionalmente por paréntesis [2].

9. Una *Condición* comparará dos *Expresiones* mediante un *Comparador* (operador de comparación numérica). Las condiciones deben ir entre paréntesis.

10. Un *Término* puede ser o bien una *Variable,* o bien un *Valor,* o bien un *Parámetro*.

11. *Operadores* válidos son: +, -, *, /, % (resto).

12. Los *Comparadores* válidos serán == (igual), != (distinto), < (menor), > (mayor).

13. Un *Parámetro* [3] se representa mediante $ seguido de un número.

14. Una *Variable* estará representada por ‘j’, ‘k’, ‘m’, ‘n’.

15. Un *Valor* será cualquier número entero mayor o igual que 0.

[1] Explicación de la Condicional if: El valor de la *Condición* determina si se debe ejecutar la *Sentencia*.
[2] Los paréntesis deben codificarse como corchetes para *jflap*.
[3] Los parámetros permiten acceder a cada uno de los campos contenidos en cada línea del fichero de entrada, y que supondremos separados por espacios entre sí. El primer campo se indexará con $1. $0 servirá para obtener la línea de entrada completa.

3. PRÁCTICA A DESARROLLAR

Teniendo en cuenta las especificaciones anteriores, se deberá diseñar una gramática que describa el lenguaje propuesto. **Dadas las limitaciones de JFLAP restringiremos el lenguaje al especificado a partir del punto 3 (descripción de Bloque).**

El procedimiento de diseño puede basarse en traducir paso a paso las especificaciones planteadas a producciones de una gramática, y en resolver pequeñas partes del Lenguaje mediante sub-gramáticas que luego pasarán a formar parte de una gramática más completa, en analogía a lo que se estila en la programación estructurada.

Por ejemplo, los dos primeros puntos dicen: "Un *Programa* se compone de uno o varios pares *Patrón Acción*, donde la *Acción* es un *Bloque* entre llaves."
y
"Un *Patrón* puede ser una etiqueta BEGIN o END, o también puede ser *lambda*."

Esto se puede traducir de la siguiente manera.

Programa → *Patron Accion*

para el caso base, y recursivamente para más de una ocurrencia:

Programa → *Patron Accion Programa*

Como se indica que Acción es un Bloque entre llaves, podríamos añadir la producción:

Accion → { *Bloque* }

Pero si lo incluimos directamente en las producciones de Programa, nos ahorramos un No Terminal.

Programa → *Patron* { *Bloque* } | *Patron* { *Bloque* } *Programa*

El segundo punto lo representamos con:

Patron → λ | BEGIN | END

Los términos BEGIN y END serían Terminales de nuestra gramática.
Considerando los no terminales P (*Programa*), B (*Bloque*) y A (*Patron*), y los Terminales b y e para representar BEGIN y END, tendríamos:

P → A{B}|A{B}P

A → λ|b|e

Este proceso debe realizarse para cada uno de los puntos de las especificaciones, obteniendo un no terminal por cada punto.

La gramática diseñada se evaluará mediante un conjunto de palabras que deberán ser aceptadas o rechazadas en el derivador CYK de *jflap*.

Más adelante encontraréis unos ejemplos que podréis utilizar para una validación preliminar. Deberéis diseñar otros ejemplos para asegurar que la gramática está bien diseñada, incluyendo palabras de no aceptación.

A continuación tenéis una serie de cuestiones técnicas a tener en cuenta para realizar la práctica.

CUESTIONES TÉCNICAS

Jflap sólo puede trabajar con símbolos de un carácter. Además, está limitado a emplear las 26 letras mayúsculas como símbolos no terminales. Esto obliga a recodificar todos los símbolos no terminales de nuestra gramática con alguna letra mayúscula. Recomendamos (siempre que sea posible) usar alguna que resulte significativa (B para *Bloque*, E para *Expresión*, etc.).

No se pueden emplear paréntesis como símbolos terminales en JFLAP, de ahí que usemos siempre corchetes.

Los símbolos terminales deben representarse con un símbolo de un único carácter. Esto afecta sobre todo a los terminales *print* e *if*, que deberemos codificar con una sola letra ("p" e "i").

Otra dificultad la encontramos en lo que se denominan separadores. Si observáis los ejemplos de programas en la introducción, veréis que casi todos los símbolos están separados por espacios. En nuestra gramática va a ser muy difícil poder incluir estos espacios. Tendremos que prescindir de ellos. Esto nos obliga a recodificar los programas en una notación manejable por *jflap.*

Programas de Prueba:

De los ejemplos anteriores…	**quedaría:**
`j=0;`	`j=0;`
`j=j+1;`	`j=j+1;`
`print j ;`	`pj;`
`print $1; print $3;`	`p$1;p$3;`
`j=j+1; if(j%2==0} print $0;`	`j=j+1;i[j%2==0]p$0;`
`j=(j+1); if ((j%2)==0) print $0;`	`j=[j+1];i[[j%2]==0]p$0;`
`print ($1 + ($2 * 43 )) ;`	`p[$1+[$2*43]];`

A la hora de derivar palabras con la gramática diseñada podréis comprobar que el algoritmo por fuerza bruta es completamente inviable por el tiempo que consume. Tendremos que recurrir a un algoritmo distinto, el derivador CYK. Como contrapartida estaremos obligados a convertir **previamente** la gramática a Forma Normal de Chomsky. **El propio algoritmo CYK puede realizar esta conversión de forma interna, pero en este caso no funciona correctamente a la hora de verificar las palabras.**

Es importante recordar que *jflap* limita los no terminales a las 26 letras en mayúscula. El proceso de bien-formar y convertir la gramática a su Forma Normal de Chomsky requiere de bastantes no terminales auxiliares, y es fácil que exceda de los 26 disponibles. Procurad diseñar una gramática lo más compacta posible. Evitad en lo posible redundancias. Vuestra gramática puede contener redenominaciones y producciones no generativas, lo cual puede ayudar a la legibilidad de la gramática. Podéis probar a sustituirlas, aunque esto ya lo realiza *jflap* automáticamente en el bien formado, por lo que no debería influir mucho.

Otro inconveniente de las gramáticas bien-formadas y, en especial, en FN de Chomsky es que pueden ser prácticamente indescifrables. Por ello, no sirve de mucho que consigáis diseñar una gramática en FN de Chomsky que funcione. **El objetivo es diseñar una gramática que se pueda bien-formar y convertir con *jflap* a FNC**.

Con una gramática bien diseñada y compacta es perfectamente posible hacer pruebas con programas con un cierto número de sentencias en un tiempo razonable. Haced las pruebas incrementando la dificultad de forma progresiva (como en los ejemplos propuestos anteriormente). Utilizad la opción de carga de ficheros de prueba (una palabra de prueba por línea) con "Load Inputs" en el "Multiple CYK Parse" para facilitar el proceso.

¿Qué hacer en caso de error?

Tal como se han planteado las especificaciones, deberíais obtener una gramática compacta que no dé problemas. Si os surge algún error, deberéis comenzar por revisar las producciones que habéis obtenido en cada punto de las especificaciones. Revisad los símbolos que habéis empleado para representar cada no terminal. Una fuente de problemas es el uso de la misma letra para distintos no terminales.

Recordad también una peculiaridad del derivador CYK en *jflap*. Si por error la palabra contiene terminales que no pertenecen al alfabeto, el derivador responderá con un "rechazo", lo cual dificultará la detección del error. Esto afecta también a ciertos símbolos como el paréntesis y el espacio en blanco. El derivador no puede reconocerlos, lo único que hace es rechazar las palabras que los contienen.

¿Qué hacer si el conversor a FNC da error por falta de no terminales?

a. Podéis intentar aplicar un proceso de factorización en alguna producción. Por ejemplo, la gramática S→aa|ab|ac tal como está planteada, requiere de 3 no terminales adicionales para pasarla a FNC (S→AA|AB|AC, A→a, B→b, C→c). Si aplicamos previamente una factorización (la primera a en la parte derecha es común a las tres producciones) obtenemos S→aA, A→a|b|c (algo parecido a sacar factor común sobre las partes similares de varias producciones), el paso a FNC (S→KA, A→a|b|c, K→a) necesitará un solo no terminal adicional, con lo que ahorramos un NT en total.

Esta factorización no deberá realizarse si en alguna de las producciones resultantes aparece λ, dado que requeriría de un bien formado con el consiguiente aumento de no terminales necesarios.

b. Hay elementos que se especifican en un punto concreto, pero que pueden ser 'desplazados' a un punto superior o inferior alterando el número de No Terminales al final de las conversiones. Ver el caso de las llaves en las especificaciones 1 y 2.

c. **En caso extremo** podéis prescindir de algún elemento como el comparador !=, o restringir el tamaño de los números a un solo dígito.

Estos cambios sólo se deberán aplicar si os faltan no terminales. Con las especificaciones dadas, esto no debería suceder, pero depende de vuestro diseño de la gramática inicial.

4. TAREAS, DOCUMENTACIÓN y ENTREGA

Grupos:

- **Esta práctica se realizará en grupos de 2 personas**. **La entrega sólo la realizará uno** de los miembros de cada grupo.

Tareas a realizar:

1. Diseño de la gramática a partir de las especificaciones. Esta tarea consiste en la elaboración de dos gramáticas equivalentes que sólo deben diferir en la nomenclatura utilizada en su codificación. La primera gramática ha de estar escrita en lenguaje natural y **se entregará en papel el día de la sesión especial** de JFLAP **(25 y 26 de Noviembre) y como fichero (G1.txt) en la entrega final.** La segunda gramática, G1.jff, debe corresponderse con la codificación para JFLAP de la gramática anterior y debe entregarse junto al resto de ficheros que se especifican en este documento. Con el objetivo de que la gramática codificada para JFLAP sea inteligible, será necesario indicar en la memoria la relación entre los símbolos no terminales usados en la codificación (letras mayúsculas) y los descriptores usados en las especificaciones. Por ejemplo, si los símbolos usados para codificar Bloque y Sentencia son B y S respectivamente, se deberá incluir explícitamente esta relación: B – Bloque, S – Sentencia.

2. Codificación para JFLAP de una gramática corregida (G2.jff). Es posible que la gramática (G1) diseñada durante la sesión especial de JFLAP sea incorrecta o que JFLAP no pueda convertirla a FNC. Si se da esta circunstancia, y durante el desarrollo de la práctica os veis obligados a realizar modificaciones, éstas deberán quedar documentadas en la memoria. **En ningún caso** se debe entregar en este apartado una gramática bien formada o convertida a FNC. De eso se encargará *JFLAP* (ver apartado 3). Para facilitar la corrección de esta gramática, en la memoria se debe incluir su transcripción a lenguaje natural y la relación existente entre los símbolos no terminales usados en la codificación y los descriptores usados en las especificaciones. En el supuesto de que no fuera necesario realizar esta tarea (por considerarse que la gramática elaborada en el apartado 1 es correcta) será necesario indicar esta circunstancia en la memoria.

3. Obtención mediante JFLAP de la gramática bien-formada y en Forma Normal de Chomsky (G3.jff) a partir de la G2.

4. Elaborar dos conjuntos de palabras para validar la gramática (G3) con el derivador CYK. Un fichero (aceptadas.txt) contendrá las palabras que la Gramática genera, y el otro (rechazadas.txt) las palabras que no pueda generar. Los ejemplos de prueba son imprescindibles para validar la práctica realizada. Dichos ejemplos deben ser exhaustivos y estar diseñados de forma meditada.

Documentos y ficheros a entregar:

- **Memoria (documento en pdf) que debe ser comprensible y explicar el trabajo realizado**. Esto incluye la G1, los pasos realizados para obtener la G2, la codificación para *jflap* de los símbolos de la gramática, etc. También se deben incluir las palabras usadas para validar el programa (aceptadas y rechazadas), los motivos por los que se ha seleccionado cada una de estas palabras y una captura legible de la respuesta dada por el derivador CYK para la gramática convertida a FNC.
- **Ficheros jff** con todas las gramáticas obtenidas y **ficheros txt** con las gramáticas G1 y G2 en lenguaje natural..
- **Dos ficheros de texto con las palabras de prueba aceptadas** y con las palabras rechazadas (una por línea) usadas en la tarea 4. Es imprescindible que estos ficheros puedan cargarse en el "Multiple CYK Parse" con la opción "Load Inputs".

Normas de entrega:

- La entrega se realizará a través de AG y antes del **13 de Diciembre de 2019 a las 23:55h.** Se adjuntará un fichero comprimido en formato **ZIP** (*JFLAP3_Apellido1a-Apellido2a_Apellido1b-Apellido2b.zip,* donde *Apellido1a-Apellido2a* son los apellidos del alumno que realiza la entrega*)* con todos los ficheros solicitados. El enlace estará en la página de grupos *reducidos*.
- La primera página del documento deberá identificar correctamente a los autores.
- Comprobad que el documento entregado se pueda abrir y que sea legible.