Lenguajes de Programación y Procesadores de Lenguaje

Grado en Ingeniería de Computadores

Profesor: Rafael del Vado Vírseda

Curso 2019-2020

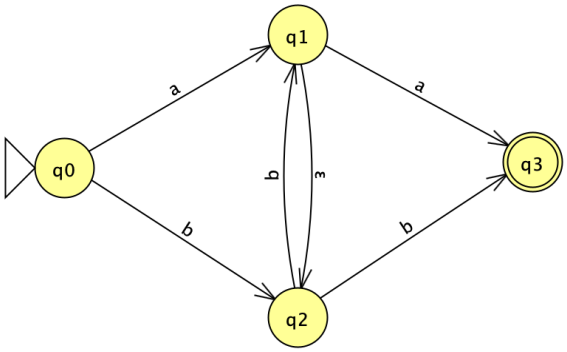
Convocatoria Extraordinaria de Septiembre

(Modalidad B: Ordenador)

Ejercicio 1 (Análisis léxico con la herramienta `JFLAP`: 1.5 puntos)

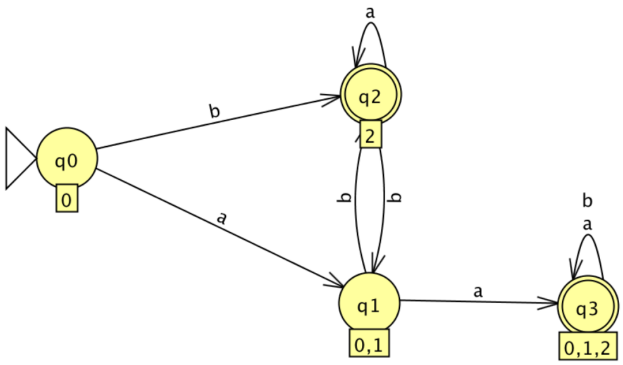
(a) **(0.5 puntos)** Usando la herramienta gráfica `JFLAP` transforma cada una de las siguientes tres expresiones regulares en un **AFD mínimo**: $((aa \,|\, b)^+ \,|\, aba)^*$, $(00 \,|\, 10^+1)?$, $(+ \,|\, -)?\, d^+(r\, d^+)?$.

(b) **(0.5 puntos)** Comprueba con la herramienta `JFLAP` si son equivalentes el autómata finito



y la expresión regular $((aa)?)^*\, bb\, (b?)^*$.

(c) **(0.5 puntos)** Considera el siguiente **AFD**:



Este **AFD** es el resultado de convertir un **AFN** tal que las etiquetas que se muestran en la figura corresponden a los estados del **AFN** original. Crea, con la ayuda de la herramienta `JFLAP`, el **AFN** original.

Ejercicio 2 (Análisis sintáctico con las herramientas `JFLAP`, `Context-Free Grammar Checker` y `Parsing Simulator`: 2.5 puntos)

(a) **(0.5 puntos)** Elimina las reglas no generativas, las reglas superfluas y los símbolos inaccesibles de la siguiente gramática usando las herramientas `JFLAP` y/o `Context-Free Grammar Checker`:

$$\begin{cases} S & \to & a\ X \\ S & \to & a \\ X & \to & b\ S \\ X & \to & b\ Y\ b\ Y \\ Y & \to & b\ a \\ Y & \to & a\ Z \\ Z & \to & a\ Z\ X \end{cases} \tag{1}$$

(b) **(0.5 puntos)** Dada la siguiente gramática:

$$\begin{cases} S & \to & x\ A\ y \\ S & \to & x\ B\ y\ y \\ S & \to & \varepsilon \\ A & \to & x\ A\ y \\ A & \to & \varepsilon \\ B & \to & x\ B\ y\ y \\ B & \to & \varepsilon \end{cases} \tag{2}$$

Calcula mediante la herramienta `JFLAP` los conjuntos de predicción `PRIMERO` y `SIGUIENTE`, y la tabla de análisis sintáctico **LL(1)** ¿Es una gramática **LL(1)**? ¿Qué tipo de conflictos presenta?

(c) **(0.75 puntos)** Considera la siguiente gramática para un **lenguaje de expresiones de tipos**:

$$\begin{cases} Tp_0 & \to & Tp_1 \Rightarrow Tp_1 \\ Tp_0 & \to & Tp_1 \\ Tp_1 & \to & Tp_2 \times Tp_1 \\ Tp_1 & \to & Tp_2 \\ Tp_2 & \to & Tp_2\,@ \\ Tp_2 & \to & Tp_3 \\ Tp_3 & \to & \textbf{int} \\ Tp_3 & \to & \textbf{float} \\ Tp_3 & \to & (\,Tp_0\,) \end{cases} \tag{3}$$

Transforma la gramática mediante la herramienta `Context-Free Grammar Checker` para obtener una gramática **LL(1)** equivalente. Obtén una derivación y el árbol sintáctico correspondiente para la entrada **int** @ $\times$ **int** $\Rightarrow$ **float** usando las herramientas `JFLAP` y `Parsing Simulator`.

(d) **(0.75 puntos)** Construyendo los autómatas que consideres oportuno mediante las herramientas `Context-Free Grammar Checker` y/o `Parsing Simulator` para la siguiente gramática

$$\begin{cases} A & \to & B \\ A & \to & C\ u \\ A & \to & v\ C \\ A & \to & v\ B\ u \\ B & \to & D \\ C & \to & D \\ D & \to & x\ D \\ D & \to & \varepsilon \end{cases} \tag{4}$$

responde justificadamente a las siguientes preguntas: ¿Admite los tipos de análisis sintáctico ascendente **LR(0)**, **SLR(1)**, **LR(1)** y/o **LALR(1)**? ¿Qué tipos de conflictos existen en cada caso? Construye una cadena de entrada de longitud 5 que se pueda reconocer mediante esta gramática, y obtén su correspondiente derivación mediante la herramienta `Parsing Simulator` usando la tabla de análisis sintáctico del analizador ascendente más eficiente que hayas encontrado. Utiliza la herramienta `JFLAP` para obtener gráficamente su árbol sintáctico correspondiente.

Ejercicio 3 (Traducción dirigida por sintaxis y análisis semántico con las herramientas `FLEX` y `BISON`: 1.5 puntos)

Considera una extensión de la gramática de la **Práctica 2** para que pueda ahora reconocer **asignaciones condicionales de expresiones aritméticas** `if_then_else`:

$$\begin{cases} \textit{calculadora} & \rightarrow & \texttt{if (}\textit{condicion}\texttt{) then \{ id =}\textit{expresion}\texttt{;\} else\{ id =}\textit{expresion}\texttt{;\}} \\ \textit{condicion} & \rightarrow & \texttt{true} \mid \texttt{false} \mid \texttt{id >=}\textit{expresion} \\ \textit{expresion} & \rightarrow & \texttt{num} \mid \texttt{id} \mid \textit{expresion} + \textit{expresion} \mid \textit{expresion} \texttt{ * } \textit{expresion} \end{cases} \tag{5}$$

Se pide:

(a) **(0.25 puntos)** Implementa un **analizador léxico** en el lenguaje de programación `C` con la herramienta `FLEX` que permita reconocer asignaciones condicionales de expresiones aritméticas.

(b) **(0.50 puntos)** Implementa un **analizador sintáctico** en el lenguaje de programación `C` con la herramienta `BISON` para poder reconocer asignaciones condicionales de expresiones aritméticas.

(c) **(0.75 puntos)** Inserta **atributos** y **acciones semánticas** para implementar con las herramientas `FLEX` y `BISON` en el lenguaje de programación `C` un **intérprete** de asignaciones condicionales de expresiones aritméticas.

Por **ejemplo**, ante la siguiente entrada, dada en un archivo de texto:

```
x = 10 ;
y = 1 ;
z = 0 ;
if ( x >= 0 ) then { y = x - 1 ; } else { z = x + 1 ; }
```

la salida por pantalla del **intérprete** pedido debería ser:

```
El valor del identificador x es 10.00
El valor del identificador y es 9.00
El valor del identificador z es 0.00
```

Ejercicio 4 (Generación de código con las herramientas `ANTLR` y `ANTLRWorks`: 2 puntos)

Considera una extensión de la gramática de la **Práctica 3** para que pueda reconocer ahora **instrucciones de selección múltiple** `switch`. Para ello, comienza añadiendo a tu práctica el siguiente código:

```
statement
    :  'switch' parExpression '{' switchBlockStatementGroups '}'
    ;

parExpression
    :   '(' expresion ')'
    ;

switchBlockStatementGroups
    :   ( switchBlockStatementGroup )*
    ;

switchBlockStatementGroup
    :
        switchLabel
        (ecuacion
        )*
    ;

switchLabel
    :   'case' expresion ':'
    |   'default' ':'
    ;
```

Se pide:

(a) **(0.75 puntos)** Implementa un **analizador léxico** y un **analizador sintáctico** en el lenguaje de programación `Java` mediante las herramientas `ANTLR` y/o `ANTLRWorks` que permita reconocer las instrucciones de selección múltiple de esta nueva gramática extendida.

(b) **(1.25 puntos)** Implementa un **compilador** en el lenguaje de programación `Java` mediante las herramientas `ANTLR` y/o `ANTLRWorks` que traduzca las instrucciones de selección múltiple de esta gramática extendida a **código de 3-direcciones**, representado mediante **cuádruplos**.

Por **ejemplo**, para la siguiente entrada:

```
switch (x) { case 1 : y = y + 1 ; case 2 : z = z * y ; default : z = x + y ; }
```

el código que debería generar como salida el **compilador** pedido sería el siguiente:

```
L01 : ( IF_TRUE, x == 1, GOTO, L03 )
L02 : ( IF_FALSE, x == 1, GOTO, L06 )
L03 : ( ADD, t0, y, 1 )
L04 : ( ASSIGN, y, t0, NULL )
L05 : ( IF_TRUE, NULL, GOTO, L14 )
L06 : ( IF_TRUE, x == 2, GOTO, L08 )
L07 : ( IF_FALSE, x == 2, GOTO, L11 )
L08 : ( MULT, t1, z, y )
L09 : ( ASSIGN, z, t1, NULL )
L10 : ( IF_TRUE, NULL, GOTO, L14 )
L11 : ( ADD, t2, x, y )
L12 : ( ASSIGN, z, t2, NULL )
L13 : ( IF_TRUE, NULL, GOTO, L14 )
L14 : ( HALT, NULL, NULL, NULL )
```