PDL – 1º PARCIAL – 23/01/2008

1. **Dada la siguiente gramática:**

   S → **(** L **)** | **a**

   L → S **,** L | S

   a) (2,5 pto) Costruid la Tabla de Análisis LALR(1) a partir de la colección canónica de conjuntos de ítems LR(1).
   b) (0,75 pto) Proporcionad la traza de análisis LALR(1) para la cadena: (a , (a))
   c) (1,25 pto) ¿Es una gramática LL(1)? Obtened una gramática LL(1) equivalente y construid su tabla de análisis LL(1).
   d) (0,5) Porporcionad la traza de análisis LL(1) para la misma cadena: (a , (a))

2. **(2 pto.) Dada la siguiente gramática que genera árboles binarios de números enteros (en forma linealizada):**

   S → A

   A → (num A A)

   A → (num)

   Definamos el nivel de profundidad de un nodo en un árbol binario como el número mínimo de ramas que hay que recorrer para ir del nodo raíz hasta este nodo. Diseñar un ETDS para esta gramática que obtenga el nivel de profundidad que ocupa el número entero de mayor valor; en caso de que el valor máximo se repita, se debe indicar la profundidad mínima. Ejemplo, en (2 (1 (0) (1)) (3 (2) (3))) la contestación sería: valor máximo 3 y profundidad 1.

Solución 1:

S → { A.niv := 0 } A { print(A.max ; A.prof)

A → **(num**

{ A_1.niv := A.niv + 1 } A_1

{ A_2.niv := A.niv + 1 } A_2 **)**

{ A.max := maximo(num.val, A_1.max, A_2.max);

Si (A.max = A_1.max) **entonces** A.prof := A_1.prof;

Si (A.max = A_2.max) **entonces** A.prof := A_2.prof;

Si (A.max = A_1.max) **y** (A.max = A_2.max) **entonces** A.prof := minimo(A_1.prof; A_2.prof)

Si (A.max = num.val) **entonces** A.prof := A.niv }

A → (num) { A.max := num.val; A.prof := A.niv }

Solución 2:

S → A { print(A.max ; A.prof)

A → **(num** A_1 A_2 **)**

{ A.max := maximo(num.val, A_1.max, A_2.max);

Si (A.max = num.val) **entonces** A.prof := 0 }

Si_No Si (A.max = A_1.max) **entonces**

Si (A.max = A_2.max) **entonces** A.prof := minimo(A_1.prof; A_2.prof) + 1

Si_No A.prof := A_1.prof + 1;

Si_No A.prof := A_2.prof + 1;

A → (num) { A.max := num.val; A.prof := 0 }

3. (1.5 pto.) Dada una gramática G que sea LL(1), indicad si las siguientes afirmaciones son ciertas o falsas (cada respuesta equivocada restará 0.25 ptos.)

a) Al construir el autómata LALR(1) para G, fusionando estados del autómata LR(1), pueden aparecer conflictos desplazamiento/reducción.

Cierto. Aunque la mayoría de gramáticas LL(1) son LALR(1), hay gramáticas LL(1) que no son LALR(1). Por lo tanto, al construir el autómata LALR(1) fusionando estados podrían aparecer conflictos.

b) G es SLR(1).

No es cierto. El hecho de ser LL(1) implica que también es LR(1), pero no tiene porqué ser SLR(1).

c) G No pude ser ambigua ni recursiva a derechas.

No es cierto. El hecho de ser LL(1) implica que no es ambigua ni recursiva a izquierdas, pero no hay nada que impida que sea recursiva derechas.

d) La tabla de análisis LR(1) de G no podrá contener ninguna columna vacía.

Es cierto. Si tuviese alguna columna vacía, al aparecer el símbolo terminal correspondiente a dicha columna en la cadena de entrada, el analizador no haría nada.

Otra forma de razonar esta cuestión es considerar que, por los algoritmos con los que se construye la tabla LR(1) (clausura y sucesor), cualquier símbolo terminal que aparezca en una producción de la gramática acabará apareciendo en algún elementos LR(1) y mediante la operación sucesor, acabará habiendo una transición etiquetada con dicho símbolo a otro estado. Dicha transición se reflejará como un desplazamiento en la tabla LR(1). La única excepción es el símbolo $, para el que no tiene porqué existir un desplazamiento pero tendrá una operación ACEPTAR.

e) La tabla de análisis LR(1) de G no podrá contener ninguna fila vacía.

Es cierto. Cada fila representa un estado del autómata LR(1). Si una fila estuviese vacía, cuando el analizador alcanzase ese estado, se pararía.

Otra forma de razonar esta cuestión es considerar que: Cada fila representa un estado del autómata LR(1). Los estados sin elementos LR(1) no se representan en la tabla. Cualquier estado no vacío tendrá algún elemento LR(1), y todo elemento LR(1) indica que se debe realizar (y por lo tanto incluir en la tabla LR(1)) un desplazamiento, una reducción, o la aceptación. Por lo tanto no puede aparecer en la tabla LR(1) ningún estado con toda la fila vacía.

f) La tabla de análisis LL(1) de G no podrá contener ninguna columna sin una acción derivar.

Es falso. En la tabla LL(1) hay una entrada para cada símbolo terminal de la gramática. Si un símbolo terminal no pertenece a PRIM(ω SIG(A)) para ningún no-terminal A, tendrá su columna vacía.

Ej. S-> a b c X

X -> a

Los símbolos terminales b y c, no pertenecen a PRIM(ω SIG(A)) para ningún A ∈ N

4. Dada la siguiente gramática, donde id representa un identificador típico de C y cte una constante entera sin signo:

```
I →   Insif   |   id = E
Insif → if ( E ) I   |   if ( E ) I else I
E → id   |   cte
```

a) (1 pto.) Escribid la especificación léxica Flex y sintáctica Bison para la gramática.
b) (0,5 pto.) Tras compilar con Bison, éste nos indica que se ha producido un conflicto en un cierto estado, cuyo contenido sabemos que es:

```
Insif → if ( E ) I .          , else / $
Insif → if ( E ) I . else     , $
```

Indicad explícitamente en la especificación sintáctica que Bison debe resolver este conflicto de forma que un "else" se anide siempre con el "if" más próximo.

Solución:

Flex:

```
%{
#include <stdio.h>
#include "asin.h"

%}
id [a-z][a-z|0-9]*
cte [0-9]*

%%
[\t \n]
if              { return(IF_); }
"("             { return(PARENT_IZQ_); }
")"             { return(PARENT_DER_); }
else            { return(ELSE_); }
"="             { return (IGUAL_); }
{cte}           { return(CTE_); }
{id}            { return(ID_); }
. {yyerror("Error lexico");}

%%
```

Bison:

```
%{
#include <stdio.h>
%}

%token IF_
%token PARENT_IZQ_
%token IGUAL_
%token ID_
%token CTE_

%left PARENT_DER_
%left ELSE_

%%
```

```
i :  inst_if
  |  ID_ IGUAL_ e
  ;
inst_if :   IF_ PARENT_IZQ_ e PARENT_DER_ i
      |     IF_ PARENT_IZQ_ e PARENT_DER_ i ELSE_ i
     ;
e :   ID_
  |   CTE_
 ;
%%
```

Obsérvese que hemos definido la precedencia del token “ELSE_” mayor que la del token “PARENT_DER”. Con esto se resuelve el conflicto reducción/desplazamiento del apartado b a favor del desplazamiento. De esta manera un “else” se asociará siempre al “if” más cercano.

Para resolver los posibles conflictos desplazamiento/reducción, Bison asigna a cada regla la precedencia del último terminal que aparece en su lado derecho. La resolución del conflicto se realiza comparando la precedencia de la regla considerada y el token de anticipación:

Si la precedencia del token es mayor que la precedencia de la regla, se selecciona la acción “desplazar”. Si la precedencia del token es menor que la precedencia de la regla, se “reduce”. En el caso de que las precedencias sean iguales, Bison considerará la asociatividad de ese nivel de precedencia.

Como la precedencia de cada símbolo viene determinada por el orden en que se define en Bison: (de menor a mayor precedencia), asignando mayor precedencia al “else” que al paréntesis derecho (PARENT_DER) asignamos mayor precedencia al “else” que a la regla, y por lo tanto se realizará el desplazamiento.