****

**Ejercicio 1.**

Para cada uno de los siguientes lenguajes, definidos sobre el alfabeto A = {a, b, c, d, e, h, x, y, z,

0, 1, 2, 3, 4 } diseñar y definir formalmente una gramática libre de contexto que lo reconozca:

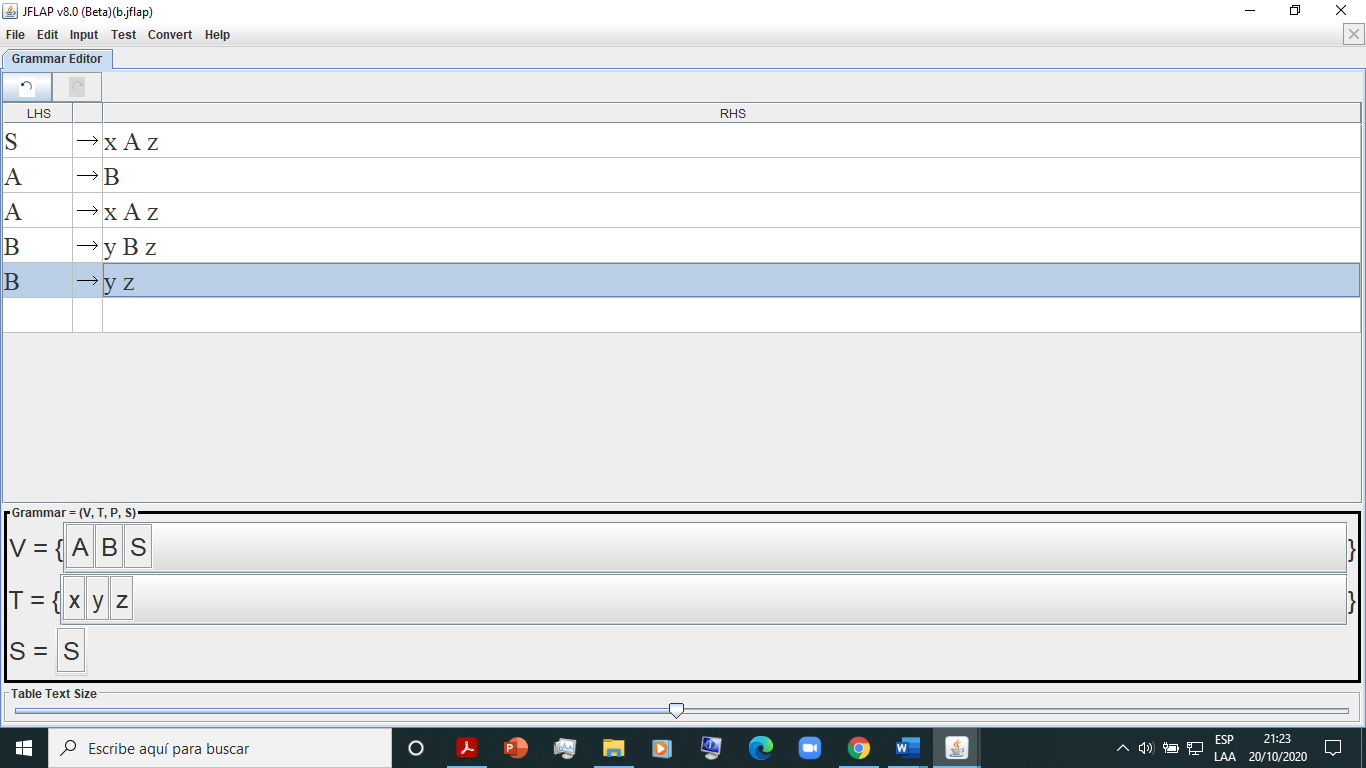
1. L2 = { xr ys zt / t = r+s y r, s ≥ 1 }

Pedido 20/10 r>=1 s>=1 t>=2 **Cadena mínima: xyzz**, Otras: xxyzzz, xyyzzz

xr ys zs zr

--B--

-------A-------



**Ejercicio 6.**

Pedido 20/10

Definir una GLC que genere cabeceras de procedimientos de la forma:

**procedure** nombre **(** lista de parámetros **)**

Donde:

* lista de parámetros será una enumeración de parámetros, separados por comas. Los parámetros tendrán la forma:

nombre tipo ent/sal [ **default** cte ]

* nombre: será una sucesión de letras y dígitos que comienza con una letra.
* tipo: puede ser char, number, string o bool
* cte: es una constante numérica o un string
* ent/sal: puede ser IN, OUT, INOUT
* los corchetes no son parte del lenguaje, indican que esa parte de la cabecera es opcional.
* Lo que figura en negrita es terminal: procedure, (, ), default

<cabecera\_Pro> ::= procedure **<nombre>** ( <**list\_param**> )<nombre> ::= <letra> / <nombre> <letra> / <nombre> <digito><letra> ::= a/b/c/.../z<digito> ::= 0/1/2/.../9

<numero> ::= <digito> / <numero> <digito><list\_param> ::= <param> / <list\_param> , <param>

<param> ::= <nombre> <tipo> <EntSal> / <nombre> <tipo> <EntSal> default <Default\_type>

<tipo> ::= char / number / string / bool

<EntSal> ::= IN / OUT /INOUT

<Default\_type> ::= ‘<ctestring>’ / <numero>

<ctestring> ::= <letra> | <ctestring> <letra> | <digito> | <ctestring> <digito> | <blanco> |

<ctestring> <blanco>

<blanco> ::= espacio en blanco

Ejemplos de hileras válidas:

- procedure mostrarlista (lista char IN,

cant number IN default 0,

salida char OUT )

- procedure vector ( vec1 string IN default ‘ ’,

vec2 string INOUT)

**Ejercicio 10.**

1. Desarrollar una gramática en formato BNF para reconocer expresiones aritméticas simples (sumas, restas, multiplicaciones y divisiones) que operan con constantes enteras e identificadores.
2. Agregar a la gramática anterior la posibilidad que la multiplicación y división tengan prioridad sobre la suma y la resta.
3. Agregar a la gramática anterior la posibilidad de incorporar expresiones entre paréntesis.

**Esta gramática es ambigua.**

E 🡪 E + E

E 🡪 E – E

E 🡪 E \* E

E 🡪 E / E

E 🡪 id | cte | (E)

No cumple el punto b. Esta gramática es ambigua.

**Gramática correcta:**

E 🡪 E + T | T

E 🡪 E – T

T 🡪 T \* F | F

T 🡪 T / F

F 🡪 id | cte | (E)

**Probar esta cadena con ambas gramáticas: id + id \* id**

**Otra cadena: (id+id) \* id**