PROCESADORES DE LENGUAJES — Primera Convocatoria

Definición del Lenguaje

IT es un lenguaje de programación secuencial basado en el uso de variables de tipo *intervalo cerrado de números enteros* (intervalo en lo siguiente). Sintácticamente, un intervalo es una expresión `[a,b]` con `a` y `b` números enteros siendo `a<=b`. Semánticamente, `[a,b]` equivale al conjunto de números enteros `{a,a+1,…,b-1,b}`. Dado un intervalo `[a,b]`, `a` se conoce como el extremo izquierdo de `[a,b]` y `b` como el extremo derecho de `[a,b]`.

El programa IT consta de una sección que declara un conjunto de variables con un valor inicial por defecto igual a `[]` (intervalo vacío) y una sección que declara secuencias de instrucciones. IT dispone de tres clases de instrucciones: (a) asignación, (b) bifurcación y (c) repetición.

La *asignación* asocia una expresión intervalo (expresión en lo siguiente) a una variable. Por ejemplo, `x = [1,3]+z;` asocia a `x` la expresión `[1,3]+z`.
La escritura de expresiones puede incluir sinónimos. IT define los siguientes sinónimos:

```
[a,b] es sinónimo de []    siendo a>b
 a es sinónimo de [a,a]    siendo a un entero cualquiera
```

Para construir expresiones, IT dispone de los operadores aritméticos: adición (`+`), diferencia (`-`) e intersección (`*`). El significado de éstos se detalla a continuación (cada intervalo no vacío en las siguientes definiciones tiene extremo izquierdo menor o igual al extremo derecho):

```
1. []+[] = [],            2. [a,b]+[] = [a,b],
3. []+[c,d] = [c,d],      4. [a,b]+[c,d] = [a+c,b+d]

5. []-[] = [],            6. []-[c,d] = [-d,-c],
7. [a,b]-[] = [a,b],      8. [a,b]-[c,d] = [a,b]+[-d,-c]

8. []*[] = [],            9. []*[c,d] = [],
10. [a,b]*[] = [],
11.  [a,b]*[c,d] = [r,s] si [r,s] es el conjunto de elementos comunes de [a,b] y [c,d]
12.  [a,b]*[c,d] = []    si [a,b] y [c,d] no tienen ningún elemento común.
```

La expresión puede usar paréntesis para explicitar asociatividades. Por ejemplo, `([1,4]+[2,5])*[0,3]`

La *bifurcación* es una construcción de IT que permite seleccionar uno de los dos bloques de instrucciones vinculados al valor de verdad de una condición. Sintácticamente, la bifurcación tiene la forma: `(cond)-> A | B` siendo `cond` la condición, `A` el bloque seleccionado si la condición es cierta y `B` el bloque seleccionado si la condición es falsa. Todo bloque de instrucciones se escribe entre llaves. Por ejemplo, `{ x=x+1; }`.

Para expresar condiciones, IT incluye operadores relacionales para construir relaciones y operadores lógicos para conectar relaciones. Los operadores relacionales en IT son : mayor (`>`), menor (`<`), mayor o igual (`>=`), menor o igual (`<=`), igual (`==`) y distinto (`!=`). La semántica de las relaciones correspondientes es la siguiente (cada intervalo no vacío en las siguientes definiciones tiene extremo izquierdo menor o igual al extremo derecho):

```
1. [] op []    es indefinido, siendo op ∈ {<,>,<=,>=}
2. [] op [c,d] es indefinido, siendo op ∈ {<,>,<=,>=}
3. [a,b] op [] es indefinido, siendo op ∈ {<,>,<=,>=}

4. [a,b]>[c,d] es cierto si a>d y falso en otro caso
5. [a,b]<[c,d] es cierto si b<c y falso en otro caso
6. [a,b]>=[c,d] es cierto si a>=d y falso en otro caso
7. [a,b]<=[c,d] es cierto si b<=c y falso en otro caso
```

8. []==[] es cierto, 9. []==[c,d] es falso,
10. [a,b]==[] es falso
11. [a,b]==[c,d] es cierto si a==c y b==d y es falso en otro caso
12. distinto (!=) es cierto/falso cuando igual (==) es falso/cierto.

Los operadores lógicos en IT son : conjunción (`&&`), disyunción (`||`) y negación (`!`) y su semántica es la siguiente:

1. relacion1 && relacion2 es indefinida si relacion1 o relacion2 son indefinidas, cierta si relacion1 y relacion2 son ciertas y falsa en las restantes situaciones.
2. relacion1 || relacion2 es indefinida si relacion1 o relacion2 son indefinidas, falsa si relacion1 y relacion2 son falsas y cierta en las restantes situaciones.
3. !relacion es indefinida si relacion es indefinida, falsa si relacion es cierta y cierta si relacion es falsa.

La *repetición* es la construcción de IT para repetir la ejecución de un bloque de instrucciones un determinado número de veces. Sintácticamente, la repetición tiene la forma: `A(numero)` donde `A` es el bloque de instrucciones y `numero` es el número de repeticiones.

A continuación, se muestra un programa IT de ejemplo con comentarios sobre su semántica):

```
PROGRAMA programa1
VARIABLES i,j,k,x,y,z;
INSTRUCCIONES
   x=[1,3];
   y=[2,4];
   z=0-[2,1];  //[0,0]-[]=[0,0]
   i=x+y;      //i=[1,3]+[2,4]=[3,7]
   k=x*y;      //k=[1,3]*[2,4]=[2,3]
   {
     (x>=1 && x<[4,6]) -> //[1,3]>=[1,1] es cierto, [1,3]<[4,6] es cierto(repeticion 1)
                          //[2,4]>=[1,1] es cierto, [2,4]<[4,6] es falso(repeticion 2)
           {x=x+1;}        //x=[1,3]+[1,1]=[2,4] (repeticion1)
         |  {x=x-[2,4];}   //x=[2,4]-[2,4]=[2,4]+[-4,-2]=[-2,2] (repeticion2)
   }(2)
   z=k+x+j; //z=[2,3]+[-2,2]+[]=[0,5]
FPROGRAMA
```

1) Análisis Léxico-Sintáctico (2.5 puntos).

SE PIDE:

1. Definición de los lexemas de IT **(1 punto)**

```
PROGRAMA:'PROGRAMA';
VARIABLES:'VARIABLES';
INSTRUCCIONES:'INSTRUCCIONES';
FPROGRAMA:'FPROGRAMA';

NUEVALINEA: '\r'?'\n';
DIGITO: [0-9];
LETRA: [a-zA-Z];

IDENT: LETRA(LETRA|DIGITO)*;
NUMERO:(DIGITO)+;

FLECHA: '->';
BARRA: '|';

Y:'&&';
O:'||';
NO:'!';

MAYOR:'>';
MENOR:'<';
MAYORIGUAL:'>=';
MENORIGUAL:'<=';
IGUAL:'==';
DISTINTO:'!=';

ADICION:'+';
DIFERENCIA:'-';
INTERSECCION:'*';

ASIG:'=';

PARENTESISABIERTO:'(';
PARENTESISCERRADO:')';
CORCHETEABIERTO:'[';
CORCHETECERRADO:']';
LLAVEABIERTA:'{';
LLAVECERRADA:'}';
PUNTOYCOMA:';';
COMA:',';
```

2. Gramática del lenguaje IT. **(1,5 puntos)**

```
programa: PROGRAMA IDENT variables instrucciones FPROGRAMA

variables: VARIABLES (vars)? PUNTOYCOMA

vars: IDENT COMA vars
    | IDENT

instrucciones: INSTRUCCIONES (instruccion)*

instruccion: asignacion | bifurcacion | repeticion

asignacion: IDENT ASIG expresion_intervalo PUNTOYCOMA

bifurcacion:
   PARENTESISABIERTO condicion PARENTESISCERRADO FLECHA bloque BARRA bloque

repeticion: bloque PARENTESISABIERTO NUMERO PARENTESISCERRADO

bloque: LLAVEABIERTA (instruccion)* LLAVECERRADA

condicion: condicion Y condicion
         | condicion O condicion
         | NO condicion
         | PARENTESISABIERTO condicion PARENTESISCERRADO
         | relacion

relacion:  expresion_intervalo MAYOR expresion_intervalo
         | expresion_intervalo MENOR expresion_intervalo
         | expresion_intervalo IGUAL expresion_intervalo
         | expresion_intervalo DISTINTO expresion_intervalo
         | expresion_intervalo MAYORIGUAL expresion_intervalo
         | expresion_intervalo MENORIGUAL expresion_intervalo

expresion_intervalo: expresion_intervalo ADICION expresion_intervalo
   | expresion_intervalo DIFERENCIA expresion_intervalo
   | expresion_intervalo INTERSECCION expresion_intervalo
   | PARENTESISABIERTO expresion_intervalo PARENTESISCERRADO
   | IDENT
   | NUMERO
   | CORCHETEABIERTO NUMERO COMA NUMERO CORCHETECERRADO
   | CORCHETEABIERTO CORCHETECERRADO
```

2) Análisis Semántico (2.5 puntos).

Supongamos la siguiente gramática para la condición del lenguaje IT:

```
condicion: condicion Y condicion
         | condicion O condicion
         | NO condicion
         | PARENTESISABIERTO condicion PARENTESISCERRADO
         | relacion

relacion:  expresion_intervalo MAYOR expresion_intervalo
         | expresion_intervalo MENOR expresion_intervalo
         | expresion_intervalo IGUAL expresion_intervalo
         | expresion_intervalo DISTINTO expresion_intervalo
         | expresion_intervalo MAYORIGUAL expresion_intervalo
         | expresion_intervalo MENORIGUAL expresion_intervalo

expresion_intervalo: expresion_intervalo ADICION expresion_intervalo
   | expresion_intervalo DIFERENCIA expresion_intervalo
   | expresion_intervalo INTERSECCION expresion_intervalo
   | PARENTESISABIERTO expresion_intervalo PARENTESISCERRADO
   | IDENT
   | NUMERO
   | CORCHETEABIERTO NUMERO COMA NUMERO CORCHETECERRADO
   | CORCHETEABIERTO CORCHETECERRADO
```

SE PIDE:

1) Decisiones y gramática atribuida para un analizador semántico capaz de decidir si una condición es indefinida suponiendo que los valores de las variables usadas en la condición se almacenan en una memoria de variables **(1,5 puntos**).

Por ejemplo, `(x>1 && x<z)` es indefinida y `(x!=z && x<y)` es definida suponiendo la memoria de variables:

Var	Valor
x	[1,3]
y	[2,4]
z	[]

```
DECISIONES
----------

1) La indefinición de una condición se calcula recursivamente sobre
   la gramática siguiendo la definición dada en el lenguaje IT (ver gramática).
   Se propone el cálculo de un atributo sintetizado (parámetro de salida cent)
   que indique si la condición es definida o indefinida.
```

Por ejemplo:

```
      condicion dev cent: cent1=condicion Y cent2=condicion
              { si cent1 o cent2 son indefinidos entonces
                   cent = indefinida
                sino
                   cent = definida
                fsi }
        | ...
```

2) La indefinición de una relación se calcula recursivamente sobre la gramática siguiendo la definición dada en el lenguaje IT (ver gramática). Se propone el cálculo de un atributo sintetizado (parámetro de salida cent) que indique si la relación es definida o indefinida.

Por ejemplo:

```
    relacion dev cent:
           v1=expresion_intervalo MAYOR v2=expresion_intervalo
           { si v1 o v2 son intervalos vacíos entonces
                cent = indefinida
             sino
                cent = definida
             fsi }
      | ...
```

3) La indefinición de una relación está íntimamente relacionada con el uso de argumentos de tipo intervalo vacío.
Para calcular el valor de una expresión intervalo, se propone el cálculo de un atributo sintetizado (parámetro de salida v).

Por ejemplo:

```
   expresion_intervalo dev v:
        v1=expresion_intervalo ADICION v2=expresion_intervalo
        { si v1 es el intervalo vacío entonces
             v=v2
          sino
             si v2 es el intervalo vacío entonces
                v = v1
             sino
                extremo izq de v = extremo izq de v1 + extremo izq de v2
                extremo der de v = extremo der de v1 + extremo der de v2
             fsi
          fsi }
```

GRAMÁTICA ATRIBUIDA

(global)
   memoria de variables
      (ejemplo)

Var	Valor
x	[1,3]
y	[2,4]
z	[]

```
condicion dev cent: cent1=condicion Y cent2=condicion
          { si cent1 o cent2 son indefinidos entonces
               cent = indefinida
            sino
               cent = definida
            fsi }
     | cent1=condicion O cent2=condicion
            { si cent1 o cent2 son indefinidos entonces
                 cent = indefinida
              sino
                 cent = definida
              fsi }
     | NO cent=condicion
     | PARENTESISABIERTO cent=condicion PARENTESISCERRADO
     | cent=relacion


 relacion dev cent:
        v1=expresion_intervalo MAYOR v2=expresion_intervalo
        { si v1 o v2 son intervalos vacíos entonces
             cent = indefinida
          sino
             cent = definida
          fsi }
   | v1=expresion_intervalo MENOR v2=expresion_intervalo
          { si v1 o v2 son intervalos vacíos entonces
              cent = indefinida
           sino
              cent = definida
           fsi }
  | v1=expresion_intervalo IGUAL v2=expresion_intervalo
        { cent = definida }
  | v1=expresion_intervalo DISTINTO v2=expresion_intervalo
        { cent = definida }
  | v1=expresion_intervalo MAYORIGUAL v2=expresion_intervalo
          { si v1 o v2 son intervalos vacíos entonces
              cent = indefinida
           sino
              cent = definida
           fsi }
  | v1=expresion_intervalo MENORIGUAL v2=expresion_intervalo
          { si v1 o v2 son intervalos vacíos entonces
              cent = indefinida
           sino
              cent = definida
           fsi }


expresion_intervalo dev v:
     v1=expresion_intervalo ADICION v2=expresion_intervalo
     { si v1 es el intervalo vacío entonces
          v=v2
       sino
          si v2 es el intervalo vacío entonces
             v = v1
          sino
             extremo izq de v = extremo izq de v1 + extremo izq de v2
             extremo der de v = extremo der de v1 + extremo der de v2
          fsi
       fsi }
   | v1=expresion_intervalo DIFERENCIA v2=expresion_intervalo
   { si v2 es el intervalo vacío entonces
        v=v1
     sino
        si v1 es el intervalo vacío entonces
           extremo izq de v = - extremo der de v2
           extremo der de v = - extremo izq de v2
```

```
     sino
        extremo izq de v = extremo izq de v1 - extremo der de v2
        extremo der de v = extremo der de v1 - extremo izq de v2
     fsi
  fsi }
| v1=expresion_intervalo INTERSECCION v2=expresion_intervalo
{ si v1 o v2 son el intervalo vacío entonces
   v=[]
sino
   si v1 y v2 no tienen ningún elemento común entonces
      v=[]
   sino
      aux = conjunto de elementos comunes en v1 y v2
      extremo izq de v = menor elemento de aux
      extremo der de v = mayor elemento de aux
   fsi
fsi }
| PARENTESISABIERTO v=expresion_intervalo PARENTESISCERRADO
| IDENT { v = consultar valor de IDENT en memoria de variables }
| NUMERO { extremo izq de v = NUMERO
           extremo der de v = NUMERO }
| CORCHETEABIERTO a:NUMERO COMA b:NUMERO CORCHETECERRADO
{ si a:NUMERO > b:NUMERO entonces
     v = []
  sino
     extremo izq de v = a:NUMERO
     extremo der de v = b:NUMERO
  fsi }
| CORCHETEABIERTO CORCHETECERRADO { v=[] }
```

2) Implementación Antlr4/Java de la regla `relacion` en la gramática atribuida propuesta en `1)`

Para facilitar esta implementación, haga uso de las siguientes etiquetas Antlr4 **(1 punto)** :

```
expresion_intervalo MAYOR expresion_intervalo          #relMayor
expresion_intervalo MENOR expresion_intervalo          #relMenor
expresion_intervalo IGUAL expresion_intervalo          #relIgual
expresion_intervalo DISTINTO expresion_intervalo       #relDistinto
expresion_intervalo MAYORIGUAL expresion_intervalo     #relMayorIgual
expresion_intervalo MENORIGUAL expresion_intervalo     #relMenorIgual
```

```
/*
  relacion dev cent:
    v1=expresion_intervalo MAYOR v2=expresion_intervalo
    { si v1 o v2 son intervalos vacíos entonces
         cent = indefinida
      sino
         cent = definida
      fsi }
      */
 @Override
 public Object visitRelMayor(Anasint.RelMayorContext ctx) {
     Boolean cent=null;
     //cent==true -> condición indefinida    cent==false -> condición definida
     List<Integer> v1,v2;
     v1=(List<Integer>)visit(ctx.expresion_intervalo(0));
     v2=(List<Integer>)visit(ctx.expresion_intervalo(1));
     if (v1.size()==0 || v2.size()==0) cent=true;
     else cent=false;
     return cent;
 }
 /*
| v1=expresion_intervalo MENOR v2=expresion_intervalo
      { si v1 o v2 son intervalos vacíos entonces
           cent = indefinida
       sino
           cent = definida
       fsi }
*/
 @Override
 public Object visitRelMenor(Anasint.RelMenorContext ctx) {
     Boolean cent=null;
     //cent==true -> condición indefinida    cent==false -> condición definida
     List<Integer> v1,v2;
     v1=(List<Integer>)visit(ctx.expresion_intervalo(0));
     v2=(List<Integer>)visit(ctx.expresion_intervalo(1));
     if (v1.size()==0 || v2.size()==0) cent=true;
     else cent=false;
     return cent;
 }
 /*
| v1=expresion_intervalo IGUAL v2=expresion_intervalo
    { cent = definida }
  */
 @Override
 public Object visitRelIgual(Anasint.RelIgualContext ctx) {
     Boolean cent=false;
     //cent==true -> condición indefinida    cent==false -> condición definida
     return cent;
 }
 /*
| v1=expresion_intervalo DISTINTO v2=expresion_intervalo
    { cent = definida }
```

```
 */
@Override
public Object visitRelDistinto(Anasint.RelDistintoContext ctx) {
    Boolean cent=false;
    //cent==true -> condición indefinida    cent==false -> condición definida
    return cent;
}
/*
| v1=expresion_intervalo MAYORIGUAL v2=expresion_intervalo
     { si v1 o v2 son intervalos vacíos entonces
         cent = indefinida
       sino
         cent = definida
       fsi }
 */
@Override
public Object visitRelMayorIgual(Anasint.RelMayorIgualContext ctx) {
    Boolean cent=null;
    //cent==true -> condición indefinida    cent==false -> condición definida
    List<Integer> v1,v2;
    v1=(List<Integer>)visit(ctx.expresion_intervalo(0));
    v2=(List<Integer>)visit(ctx.expresion_intervalo(1));
    if (v1.size()==0 || v2.size()==0) cent=true;
    else cent=false;
    return cent;
}
/*
| v1=expresion_intervalo MENORIGUAL v2=expresion_intervalo
     { si v1 o v2 son intervalos vacíos entonces
         cent = indefinida
       sino
         cent = definida
       fsi }
 */
@Override
public Object visitRelMenorIgual(Anasint.RelMenorIgualContext ctx) {
    Boolean cent=null;
    //cent==true -> condición indefinida    cent==false -> condición definida
    List<Integer> v1,v2;
    v1=(List<Integer>)visit(ctx.expresion_intervalo(0));
    v2=(List<Integer>)visit(ctx.expresion_intervalo(1));
    if (v1.size()==0 || v2.size()==0) cent=true;
    else cent=false;
    return cent;
}
```

3) Intérprete (2.5 puntos).

Intérprete de IT sin bifurcación. Supongamos la siguiente gramática para dicho sublenguaje:

```
programa: PROGRAMA IDENT variables instrucciones FPROGRAMA

variables: VARIABLES (vars)? PUNTOYCOMA

vars: IDENT COMA vars
    | IDENT

instrucciones: INSTRUCCIONES (instruccion)*

instruccion: asignacion | repeticion

asignacion: IDENT ASIG expresion_intervalo PUNTOYCOMA

repeticion: bloque PARENTESISABIERTO NUMERO PARENTESISCERRADO

bloque: LLAVEABIERTA (instruccion)* LLAVECERRADA


expresion_intervalo: expresion_intervalo ADICION expresion_intervalo
   | expresion_intervalo DIFERENCIA expresion_intervalo
   | expresion_intervalo INTERSECCION expresion_intervalo
   | PARENTESISABIERTO expresion_intervalo PARENTESISCERRADO
   | IDENT
   | NUMERO
   | CORCHETEABIERTO NUMERO COMA NUMERO CORCHETECERRADO
   | CORCHETEABIERTO CORCHETECERRADO
```

SE PIDE:

Decisiones y gramática atribuida para el intérprete propuesto.

```
DECISIONES
----------


 1) Se necesita una memoria (llamada p.e. memoria_variables) para almacenar
    cada variable del programa con su valor.   Ejemplo memoria_variables:

                 Var |  Valor
                 ----------------
                | x  |  [1,3]  |
                | y  |  [2,4]  |
                | z  |  []     |
                ----------------

 2) La declaración de variables inicializa memoria_variables con tales variables
    inicializadas al valor [] (intervalo vacío).

    vars: IDENT { almacenar (IDENT,[]) en memoria_variables } COMA vars
        | IDENT { almacenar (IDENT,[]) en memoria_variables }

 3) El cálculo del valor de una expresión intervalo se ha resuelto en la
    pregunta anterior.
```

4) La instrucción de asignación supone una actualización de la memoria
   de variables para la variable asignada.

   asignacion: IDENT ASIG v=expresion_intervalo PUNTOYCOMA
          { actualizar memoria_variables con (IDENT,v) }

5) Las instrucción de repetición implica interpretar
   el bloque afectado el número de veces establecido.

   repetición: bloque PARENTESISABIERTO NUMERO PARENTESISCERRADO
      { repetir NUMERO veces la interpretación de bloque }

   bloque: LLAVEABIERTA (instruccion)* LLAVECERRADA

GRAMÁTICA ATRIBUIDA

(global)
   memoria de variables
      (ejemplo)

Var	Valor
x	[1,3]
y	[2,4]
z	[]

```
programa: PROGRAMA IDENT variables instrucciones FPROGRAMA

variables: VARIABLES (vars)? PUNTOYCOMA

vars: IDENT COMA vars
    | IDENT

vars: IDENT { almacenar (IDENT,[]) en memoria_variables } COMA vars
    | IDENT { almacenar (IDENT,[]) en memoria_variables }

instrucciones: INSTRUCCIONES (instruccion)*

instruccion : asignacion | repeticion

asignacion: IDENT ASIG v=expresion_intervalo PUNTOYCOMA
  { actualizar memoria_variables con (IDENT,v) }

repeticion: bloque PARENTESISABIERTO NUMERO PARENTESISCERRADO
  { repetir NUMERO veces la interpretación de bloque }

bloque: LLAVEABIERTA (instruccion)* LLAVECERRADA

expresion_intervalo dev v:
        v1=expresion_intervalo ADICION v2=expresion_intervalo
        { si v1 es el intervalo vacío entonces
             v=v2
          sino
             si v2 es el intervalo vacío entonces
                v = v1
             sino
                extremo izq de v = extremo izq de v1 + extremo izq de v2
                extremo der de v = extremo der de v1 + extremo der de v2
             fsi
          fsi }
      | v1=expresion_intervalo DIFERENCIA v2=expresion_intervalo
      { si v2 es el intervalo vacío entonces
           v=v1
```

```
  sino
     si v1 es el intervalo vacío entonces
        extremo izq de v = - extremo der de v2
        extremo der de v = - extremo izq de v2
     sino
        extremo izq de v = extremo izq de v1 - extremo der de v2
        extremo der de v = extremo der de v1 - extremo izq de v2
     fsi
  fsi }
| v1=expresion_intervalo INTERSECCION v2=expresion_intervalo
{ si v1 o v2 son el intervalo vacío entonces
   v=[]
sino
   si v1 y v2 no tienen ningún elemento común entonces
      v=[]
   sino
      aux = conjunto de elementos comunes en v1 y v2
      extremo izq de v = menor elemento de aux
      extremo der de v = mayor elemento de aux
   fsi
fsi }
| PARENTESISABIERTO v=expresion_intervalo PARENTESISCERRADO
| IDENT { v = consultar valor de IDENT en memoria de variables }
| NUMERO { extremo izq de v = NUMERO
           extremo der de v = NUMERO }
| CORCHETEABIERTO a:NUMERO COMA b:NUMERO CORCHETECERRADO
{ si a:NUMERO > b:NUMERO entonces
     v = []
  sino
     extremo izq de v = a:NUMERO
     extremo der de v = b:NUMERO
  fsi }
| CORCHETEABIERTO CORCHETECERRADO { v=[] }
```

4) Compilador (2.5 puntos).

Compilador Java de IT sin bifurcación ni repetición. Supongamos la siguiente gramática para dicho sublenguaje:

```
programa: PROGRAMA IDENT variables instrucciones FPROGRAMA

variables: VARIABLES (vars)? PUNTOYCOMA

vars: IDENT COMA vars
    | IDENT

instrucciones: INSTRUCCIONES (instruccion)*

instruccion: asignacion

asignacion: IDENT ASIG expresion_intervalo PUNTOYCOMA


expresion_intervalo: expresion_intervalo ADICION expresion_intervalo
   | expresion_intervalo DIFERENCIA expresion_intervalo
   | expresion_intervalo INTERSECCION expresion_intervalo
   | PARENTESISABIERTO expresion_intervalo PARENTESISCERRADO
   | IDENT
   | NUMERO
   | CORCHETEABIERTO NUMERO COMA NUMERO CORCHETECERRADO
   | CORCHETEABIERTO CORCHETECERRADO
```

Supongamos las siguientes decisiones para construir el compilador:

```
DECISIONES
----------

   1) El programa IT se traduce en un main Java.
      Ejemplo:
      PROGRAMA programa1 ...

         se traduce a:

      import java.util.*;
      public class Programa1{
         public static void main(String[] args){
            ...

   2) Las variables en IT se traducen a variables en Java de tipo List<Integer>. De
      esta forma, el intervalo se traduce a una lista de enteros.


   3) De acuerdo con la definición del lenguaje IT, la declaración de variables se
      inicializa al intervalo vacío, en nuestra traducción, a la lista vacía.
      Ejemplo:
               VARIABLES x,y,z;

                  se traduce a:

               List<Integer>x=new LinkedList<>();
               List<Integer>y=new LinkedList<>();
               List<Integer>z=new LinkedList<>();
```

4) Los intervalos de la forma [a,b] no se pueden escribir directamente en Java. La manera de simularlo es a través de variables frescas (no existentes en el programa) de tipo List<Integer>. Para generar estas variables, se necesita un contador de variables frescas. El nombre de estas variables se genera con un prefijo _aux_ seguido del número almacenado en el contador. El contador se irá incrementando a conforme se declaren nuevas variables frescas:

Ejemplo: [1,3] se traduce a:

```
List<Integer>_aux_1=new LinkedList<>();
for(Integer _i=1;_i<=3;_i++)
   _aux_1.add(_i);
```

El intervalo de tipo número entero n a se trata como el intervalo [n,n].

El intervalo tipo variable x se traduce como la propia variable x.

El intervalo entre paréntesis se traduce como la variable correspondiente a dicho intervalo.
Ejemplo: ([1,3]) se traduce como _aux_2 siendo _aux_2 la variable fresca correspondiente a [1,3]

La adición de intervalos se traduce a un código que almacena el valor resultante en una variable fresca.

Ejemplo:
x+y

se traduce a:

```
List<Integer>_aux_3=new LinkedList<>();
if (x.size()==0)   _aux_3.addAll(y);
else if (y.size()==0) _aux_3.addAll(x);
     else
        for(Integer _i=x.get(0)+y.get(0);
                    _i<=x.get(x.size()-1)+y.get(y.size()-1);_i++)
           _aux_3.add(_i);
```

siendo _aux_3 la variable fresca correspondiente a la expresión x+y.

La diferencia de intervalos se traduce a un código que almacena el valor resultante en una variable fresca.

Ejemplo: x-y se traduce a:

```
List<Integer>_aux_4=new LinkedList<>();
if (y.size()==0)   _aux_4.addAll(x);
else if (x.size()==0)
       for(Integer _i=-y.get(y.size()-1);_i<=-y.get(0);_i++)
          _aux_4.add(_i);
     else
        for(Integer _i=x.get(0)+(-y.get(y.size()-1));
                    _i<=x.get(x.size()-1)+(-y.get(0));_i++)
           _aux_4.add(_i);
```

siendo _aux_4 la variable fresca correspondiente a la expresión x-y.

La intersección de intervalos se traduce a un código que almacena el valor resultante en una variable fresca.

Ejemplo: x*y se traduce a:

```
List<Integer>_aux_5=new LinkedList<>();
if (x.size()==0 || y.size()==0)   ;
else
   for(Integer _i:x)
      if (y.contains(_i))   _aux_5.add(_i);
```

siendo _aux_5 la variable fresca correspondiente a la expresión x*y.

SE PIDE:

1. Compresión del problema. Escriba el código Java correspondiente a la asignación IT `z=k+x+j`. **(1 punto)**

```
List<Integer>_aux_7=new LinkedList<>();
if (k.size()==0)   _aux_7.addAll(x);
else if (x.size()==0) _aux_7.addAll(k);
     else
         for(Integer _i=k.get(0)+x.get(0);_i<=k.get(k.size()-1)+x.get(x.size()-1);_i++)
             _aux_7.add(_i);
List<Integer>_aux_8=new LinkedList<>();
if (_aux_7.size()==0)   _aux_8.addAll(j);
else if (j.size()==0) _aux_8.addAll(_aux_7);
     else
         for(Integer _i=_aux_7.get(0)+j.get(0);_i<=_aux_7.get(_aux_7.size()-
1)+j.get(j.size()-1);_i++)
             _aux_8.add(_i);
z=_aux_8;
```

2. Gramática atribuida para el compilador propuesto. **(1.5 puntos)**

```
GRAMÁTICA ATRIBUIDA
-------------------
(global)
     indents
     contador_variables_frescas
     fichero


     abrir_fichero(nombre_clase) dev fichero
     cerrar_fichero(fichero)

     generar_codigo_cabecera_clase(nombre_clase){
        indents=0;
        escribir en fichero el texto (con indents):
             "import java.util.*;
             public class "+nombre_clase+"{"
        indents=indents+3;
     }

     generar_codigo_cabecera_main(){
        escribir en fichero el texto (con indents):"public static void main(String[]
args){"
        indents=indents+3;
     }

     generar_codigo_fin_clase(){
        indents=indents-3;
        escribir en fichero el texto (con indents): "}"
     }
```

```
     generar_codigo_fin_main(){
        indents=indents-3;
        escribir en fichero el texto (con indents): "}"
     }

     generar_codigo_declaración_variables(conjunto){
         para cada v en conjunto
           escribir en fichero el texto (con indents): "List<Integer>"+v+"=new
LinkedList<>();"
      }

      gencodigo_adicion(cod1,cod2) dev cod{
         cod="_aux_"+contador_variables_frescas
         incrementar contador_variables_frescas
         escribir en fichero el texto  (con indents): "List<Integer>"+cod+"=new
LinkedList<>();"
         escribir en fichero el texto  (con indents): "if ("+cod1+".size()==0)
"+cod+".addAll("+cod2+");"
         escribir en fichero el texto  (con indents): "else if ("+cod2+".size()==0)
"+cod+".addAll("+cod1+");"
         escribir en fichero el texto  (con indents): "     else"
         escribir en fichero el texto  (con indents): "        for(Integer
_i="+cod1+".get(0)+"+cod2+".get(0);_i<="+cod1+".get("+cod1+".size()-
1)+"+cod2+".get("+cod2+".size()-1);_i++)"
         escribir en fichero el texto  (con indents): "           "+cod+".add(_i);"
         devolver cod
      }

      gencodigo_diferencia(cod1,cod2) dev cod{
         cod="_aux_"+contador_variables_frescas
         incrementar contador_variables_frescas
         escribir en fichero el texto  (con indents): "List<Integer>"+cod+"=new
LinkedList<>();"
         escribir en fichero el texto  (con indents): "if ("+cod2+".size()==0)
"+cod+".addAll("+cod1+");"
         escribir en fichero el texto  (con indents): "else if ("+cod1+".size()==0)"
         escribir en fichero el texto  (con indents): "        for(Integer _i=-
"+cod2+".get("+cod2+".size()-1);_i<=-"+cod2+".get(0);_i++)"
         escribir en fichero el texto  (con indents): "           "+cod+".add(_i);"
         escribir en fichero el texto  (con indents): "        else"
         escribir en fichero el texto  (con indents): "           for(Integer
_i="+cod1+".get(0)+(-"+cod2+".get("+cod2+".size()-1));_i<="+cod1+".get("+cod1+".size()-
1)+(-"+cod2+".get(0));_i++)"
         escribir en fichero el texto  (con indents): "              "+cod+".add(_i);"
         devolver cod
      }

      gencodigo_interseccion(cod1,cod2) dev cod{
          cod="_aux_"+contador_variables_frescas
          incrementar contador_variables_frescas
          escribir en fichero el texto (con indents): "List<Integer>"+cod+"=new
LinkedList<>();"
          escribir en fichero el texto (con indents): "if ("+cod1+".size()==0 ||
"+cod2+".size()==0)   ;"
          escribir en fichero el texto (con indents): "else"
          escribir en fichero el texto (con indents): "   for(Integer _i:"+cod1+")"
          escribir en fichero el texto (con indents): "      if ("+cod2+".contains(_i))
"+cod+".add(_i);"
          devolver cod
      }

      gencodigo_numero(numero) dev cod{
         cod="_aux_"+contador_variables_frescas
         incrementar contador_variables_frescas
         escribir en fichero el texto (con indents): "List<Integer>"+cod+"=new
LinkedList<>();"
         escribir en fichero el texto (con indents):  cod+".add("+numero+");"
```

```
         devolver cod
      }

      gencodigo_intervalo_vacio() dev cod{
          cod="_aux_"+contador_variables_frescas
          incrementar contador_variables_frescas
          escribir en fichero el texto (con indents): "List<Integer>"+cod+"=new
LinkedList<>();"
          devolver cod
      }

      gencodigo_intervalo_no_vacio(extremo_izquierdo,extremo_derecho) dev cod{
           cod="_aux_"+contador_variables_frescas
           incrementar contador_variables_frescas
           escribir en fichero el texto (con indents): "List<Integer>"+cod+"=new
LinkedList<>();"
           escribir en fichero el texto (con indents): "for(Integer
_i="+extremo_izquierdo+";_i<="+extremo_derecho+";_i++)"
           escribir en fichero el texto (con indents): "   "+cod+".add(_i);"
           devolver cod
      }

programa: { fichero = abrir_fichero(nombre_clase)
            generar_codigo_cabecera_clase(nombre_clase) }
            PROGRAMA IDENT
            { generar_codigo_cabecera_main()
              inicializar contador_variables_frescas }
            variables instrucciones
            { generar_codigo_fin_main() }
            FPROGRAMA
            { generar_codigo_fin_clase()
              cerrar_fichero(fichero) }

variables: VARIABLES (s=vars {generar_codigo_declaracion_variables(s)})? PUNTOYCOMA

vars dev conjunto: IDENT {almacenar IDENT en conjunto} COMA aux=vars {almacenar aux en
conjunto}
    | IDENT {almacenar IDENT en conjunto}

instrucciones: INSTRUCCIONES (instruccion)*

instruccion: asignacion

asignacion: IDENT ASIG cod=expresion_intervalo PUNTOYCOMA
   {  escribir en fichero el texto (con indents):IDENT+"="+cod+";" }

expresion_intervalo dev cod:
           cod1=expresion_intervalo ADICION cod2=expresion_intervalo
           { cod=gencodigo_adicion(cod1,cod2) }
         | intervalo1=expresion_intervalo DIFERENCIA intervalo2=expresion_intervalo
           { cod=gencodigo_diferencia(cod1,cod2) }
         | intervalo1=expresion_intervalo INTERSECCION intervalo2=expresion_intervalo
           { cod=gencodigo_interseccion(cod1,cod2) }
         | PARENTESISABIERTO cod=expresion_intervalo PARENTESISCERRADO
         | IDENT  { cod=IDENT }
         | NUMERO { cod=gencodigo_numero(NUMERO) }
         | CORCHETEABIERTO a:NUMERO COMA b:NUMERO CORCHETECERRADO
            { cod=gencodigo_intervalo_no_vacio(a:NUMERO,b:NUMERO) }
         | CORCHETEABIERTO CORCHETECERRADO { cod=gencodigo_intervalo_vacio() }
```