Procesadores de lenguajes

Ejercicios del Tema 5

Ejercicio 5.1

SOLUCIÓN:

Atributos:

- Lista: s (sintetizado) (ListElement)
- SigueLista: s (sintetizado) (ListElement)

Acciones semánticas:

Lista → **num** *SigueLista*
{ Lista.s = new ListElement(Double.parseDouble(num.lexema));
Lista.s.next = SigueLista.s; }

SigueLista → **coma num** *SigueLista1*
{ SigueLista.s = new ListElement(Double.parseDouble(num.lexema));
SigueLista.s.next = SigueLista1.s; }

SigueLista → λ { SigueLista.s = null; }

Ejercicio 5.2

SOLUCIÓN:

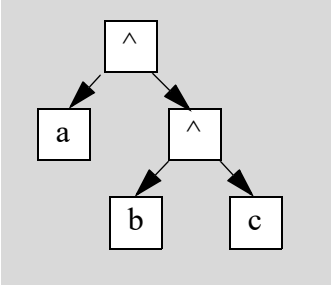
Atributos:

- Factor: s (sintetizado) (Expression)
- Potencia: h (heredado) (Number), s (sintetizado) (Expression)

Acciones semánticas:

Factor → **num** { Potencia.h = new Number(num.lexema); } *Potencia*
{ Factor.s = Potencia.s; }
Potencia → **elev** **num** { Potencia1.h = new Number(num.lexema); }
Potencia1 { Potencia.s = new Power(Potencia.h, Potencia1.s); }
Potencia → λ { Potencia.s = Potencia.h; }

Estructura para "a ^ b ^ c":



Ejercicio 5.3

SOLUCIÓN:

Atributos:

- List: first (sintetizado) (DoubleListElement)

Acciones semánticas:

```
DoubleListElement List() :
{ DoubleListElement e, first, last;}
{
   e = Element()  { first = e; last = e; }
  ( <COMMA>  e = Element()  { last.next = e; e.prev = last; last = e;} )*
  { return first; }
}
```

Ejercicio 5.4

SOLUCIÓN:

Atributos:

- Expr: s (sintetizado) (OptionList)
- Option: s (sintetizado) (ConcatList)
- Base: s (sintetizado) (Expression)
- Oper: h (heredado) (Expression), s (sintetizado) (Expression)

Acciones semánticas:

Expr → *Option* { Expr.s = new OptionList(Option.s); }

Expr → *Expr1* **or** *Option* { Expr1.s.addOption(Option.s); Expr.s = Expr1.s; }

Option → *Base* { Option.s = new ConcatList(Base.s); }

Option → *Option1* *Base* { Option1.s.concat(Base.s); Option.s = Option1.s; }

Base → **symbol** { Base.s = new Symbol(symbol.lexema); }

Base → **lparen** *Expr* **rparen** { Oper.h = Expr.s; } *Oper* { Base.s = Oper.s; }

Oper → λ { Oper.s = Oper.h; }

Oper → **star** { Oper.s = new Operation(Operation.STAR, Oper.h); }

Oper → **plus** { Oper.s = new Operation(Operation.PLUS, Oper.h); }

Oper → **hook** { Oper.s = new Operation(Operation.HOOK, Oper.h); }

Ejercicio 5.5

SOLUCIÓN:

Atributos:

- Expr: s (sintetizado) (Expression)
- Option: s (sintetizado) (Expression)
- Base: s (sintetizado) (Expression)
- Oper: h (heredado) (Expression), s (sintetizado) (Expression)

Acciones semánticas:

```
Expression Expr() :
{
 OptionList Expr_s;
 Expression Option_s;
}
{
  Option_s = Option()
  { Expr_s = new OptionList( Option_s ); }
  (
    <OR>  Option_s = Option()
    { Expr_s.addOption( Option_s); }
  )*

  { return Expr_s; }
}

Expression Option() :
{
 ConcatList Option_s;
 Expression Base_s;
}
{
 Base_s = Base()
 { Option_s = new ConcatList(Base_s); }
 (
   Base_s = Base()
   { Option_s.concat(Base_s); }
 )*

 { return Option_s; }
}
```

```
Expression Base() :
{
  Token tk;
  Expression Base_s;
  Expression Expr_s, Oper_s;
}
{
(
  tk = <SYMBOL> { Base_s = new Symbol(tk.image.charAt(0)); }
|
  <LPAREN> Expr_s = Expr() <RPAREN> Oper_s = Oper(Expr_s)
 { Base_s = Oper_s; }
)

  { return Base_s; }
}

Expression Oper(Expression Oper_h):
{
  Expression Oper_s = Oper_h;
}
{
 (
    <STAR> { Oper_s = new Operation( Operation.STAR, Oper_h ); }
  | <PLUS> { Oper_s = new Operation( Operation.PLUS, Oper_h ); }
  | <HOOK> { Oper_s = new Operation( Operation.HOOK, Oper_h ); }
 )?

  { return Oper_s;}
}
```

Ejercicio 5.6

Ejercicio 5.7

SOLUCIÓN:

Atributos:

- Árbol: s (sintetizado) (Arbol)
- Nodo: s (sintetizado) (Nodo)
- NodoNoTerminal: s (sintetizado) (Nodo)
- NodoTerminal: s (sintetizado) (Nodo)
- ListaDeCasos: h (heredado)(String), s (sintetizado) (Pregunta)
- Caso: h (heredado)(String), s (sintetizado) (Pregunta)
- Atributo: s (sintetizado) (String)
- Clase: s (sintetizado) (String)
- Valor: s (sintetizado) (int)

Acciones semánticas:

Árbol → **tree id** "**{**" *Nodo* "**}**" { Arbol.s = new Arbol(id, Nodo.s); }

Nodo → *NodoNoTerminal* { Nodo.s = NodoNoTerminal.s, }
Nodo → *NodoTerminal* { Nodo.s = NodoTerminal. s; }

NodoNoTerminal → **switch** *Atributo* "**{**"
{ ListaDeCasos.h = Atributo.s; }
ListaDeCasos "**}**"
{ NodoNoTerminal.s = ListaDeCasos.s, }

NodoTerminal → **class** *Clase* "**;**"
{ NodoTerminal.s = new Clase(Clase.s); }

ListaDeCasos → { Caso.h = ListaDeCasos.h; } *Caso* *ListaDeCasos1*
{ Caso.s.asignaFalso(ListaDeCasos1.s);
ListaDeCasos.s = Caso.s;}
ListaDeCasos → λ { ListaDeCasos.s = null; }

Caso → **case** *Valor* "**:**" *Nodo*
{ Caso.s = new Pregunta(Caso.h, Valor.s);
Caso.s.asignaCierto(Nodo.s); }

Atributo → **id** { Atributo.s = id.lexema; }
Clase → **id** { Clase.s = id.lexema; }
Valor → **num** { Valor.s = Integer.parseInt(num.lexema); }

Ejercicio 5.8

SOLUCIÓN:

Atributos:

- Circuito: s (sintetizado) (Nodo)
- CircuitoSerie: s (sintetizado) (Nodo)
- RamaParalela: h (heredado) (Nodo), s (sintetizado) (Nodo)
- CircuitoBase: s(sintetizado) (Nodo)
- ConexionSerie: h (heredado) (Nodo), s (sintetizado) (Nodo)

Acciones semánticas:

Circuito → *CircuitoSerie* {RamaParalela.h = CircuitoSerie.s; }
RamaParalela {Circuito.s = RamaParalela.s; }

RamaParalela → "|" *CircuitoSerie* { Paralelo nodo = new Paralelo();
nodo.p1 = RamaParalela.h;
nodo.p2 = CircuitoSerie.s;
RamaParalela1.h = nodo; }
RamaParalela1 { RamaParalela.s = RamaParalela1.s; }

RamaParalela → λ { RamaParalela.s = RamaParalela.h; }

CircuitoSerie → *CircuitoBase* {ConexiónSerie.h = CircuitoBase.s; }
ConexionSerie { CircuitoSerie.s = ConexiónSerie.s; }

ConexionSerie → "**-**" *CircuitoBase* { Serie nodo = new Serie();
nodo.s1 = ConexiónSerie.h;
nodo.s2 = CircuitoBase.s;
ConexiónSerie1.h = nodo; }
ConexionSerie1 { ConexiónSerie.s = ConexiónSerie1.s; }

ConexionSerie → λ { ConexiónSerie.s = ConexiónSerie.h; }

CircuitoBase → **resistencia**
{ Resistencia nodo = new Resistencia();
nodo.R = resistencia.lexema;
CircuitoBase.s = nodo; }

CircuitoBase → "**(**" *Circuito* "**)**"
{ CircuitoBase.s = Circuito.s; }

Ejercicio 5.9

Ejercicio 5.10

SOLUCIÓN:

Atributos:

- Figura: objeto (sintetizado) (Figura)
- Dimensión: width (sintetizado) y height (sintetizado)
- Particion: width (heredado), height (heredado) y objeto (sintetizado) (Particion)
- Horizontal: width (heredado), height (heredado) y objeto (sintetizado) (Horizontal)
- Vertical: width (heredado), height (heredado) y objeto (sintetizado) (Vertical)
- Color: width (heredado), height (heredado) y objeto (sintetizado) (Rectangulo)
- Factor: valor (sintetizado)

Acciones semánticas:

Figura → **figura** Dimensión
{ Partición.width = Dimensión.width;
Partición.height = Dimensión.height; }
Partición
{ Figura.objeto = new Figura(Dimensión.width,
Dimensión.height,
Partición.objeto); }

Dimensión → "**[**" **entero** "**,**" **entero** "**]**"
{ Dimensión.width = entero1.valor;
Dimensión.height = entero2.valor; }

Partición → { Horizontal.width = Partición.width;
Horizontal.height = Partición.height; }
Horizontal
{ Partición.objeto = Horizontal.objeto; }

Partición → { Vertical.width = Partición.width;
Vertical.height = Partición.height; }
Vertical
{ Partición.objeto = Vertical.objeto; }

Partición → { Color.width = Partición.width;
Color.height = Partición.height; }
Color
{ Partición.objeto = Color.objeto; }

Horizontal → **horizontal** *Factor* "**{**"
{ Partición1.width = Horizontal.width * Factor.valor;
Partición1.height = Horizontal.height; }
Partición1 "**,**"
{ Partición2.width = Horizontal.width * (1 -Factor.valor);
Partición2.height = Horizontal.height; }
Partición2 "**}**"
{ Horizontal.objeto = new Horizontal(Partición1.objeto,
Partición2.objeto); }

Vertical → **vertical** *Factor* "**{**"
{ Partición1.width = Vertical.width;
Partición1.height = Vertical.height * Factor.valor; }
Partición1 "**,**"
{ Partición2.width = Vertical.width;
Partición2.height = Vertical.height * (1 - Factor.valor); }
Partición2 "**}**"
{ Vertical.objeto = new Vertical(Partición1.objeto,
Partición2.objeto); }

Color → **color** "**(**" **entero1** "**,**" **entero2** "**,**" **entero3** "**)**"
{ Color.objeto = new Rectangulo(Color.width,
Color.height,
entero1.valor,
entero2.valor,
entero3.valor); }

Factor → "**[**" **real** "**]**"
{ Factor.valor = real.valor; }

Ejercicio 5.11

SOLUCIÓN:

Atributos:

- Escena: escena (sintetizado) (Scene)
- ListaDePuntos: esena (heredado) (Scene)
- ListaDeFiguras: escena (heredado) (Scene)
- Punto: punto (sintetizado) (Point)
- Figura: escena (heredado) (Scene), figura (sintetizado) (Figure)
- Línea: escena (heredado) (Scene), linea (sintetizado) (Line)
- Rectángulo: escena (heredado) (Scene), rectangulo (sintetizado) (Rectangle)
- Polígono: escena (heredado) (Scene), poligono (sintetizado) (Polygon)
- ContinuaListaDepuntos: escena (heredado) (Scene), poligono (heredado) (Polygon)

Acciones semánticas:

Escena → **scene llaveab**

```
{ Escena.escena = new Scene();
  ListaDePuntos.escena = Escena.escena; }
```

ListaDePuntos

```
{ ListaDeFiguras.escena = Escena.escena; }
```

ListaDeFiguras **llavece**

ListaDePuntos → *Punto*

```
{ ListaDePuntos.escena.addPoint( Punto.punto );
  ListaDePuntos1.escena = ListaDePuntos.escena; }
```

ListaDePuntos1

ListaDePuntos → λ

Punto → **point id parab num1 coma num2 parce pyc**

```
{ Punto.punto = new Point(id.lexema,
                          num1.valor, num2.valor); }
```

ListaDeFiguras →

```
{ Figura.escena = ListaDeFiguras.escena; }
```

Figura

```
{ ListaDeFiguras.escena.addFigure(Figura.figura);
  ListaDeFiguras1.escena = ListaDeFiguras.escena; }
```

ListaDeFiguras1

ListaDeFiguras → λ

Figura →

```
{ Linea.escena = Figura.escena; }
```

Línea

```
{ Figura.figura = Linea.linea; }
```

Figura →

```
{ Rectangulo.escena = Figura.escena; }
```

Rectángulo

```
{ Figura.figura = Rectangulo.rectangulo; }
```

Figura →

```
{ Poligono.escena = Figura.escena; }
```

Polígono

```
{ Figura.figura = Poligono.poligono; }
```

Línea → **line parab id1 coma id2 parce pyc**

```
{ Point p1 = Linea.escena.searchPoint(id1.lexema);
  Point p2 = Linea.escena.searchPoint(id2.lexema);
  Linea.linea = new Line(p1,p2); }
```

Rectángulo → **rectangle parab id1 coma id2 parce pyc**

```
{ Point p1 = Rectangulo.escena.searchPoint(id1.lexema);
  Point p2 = Rectangulo.escena.searchPoint(id2.lexema);
  Rectangulo.rectangulo = new Rectangle(p1,p2); }
```

Polígono → **polygon parab id**

```
{ Point p1 = Poligono.escena.searchPoint(id.lexema);
  Poligono.poligono = new Polygon(p1);
  ContinuaListaDePuntos.escena = Poligono.escena;
  ContinuaListaDePuntos.poligono = Poligono.poligono; }
```

ContinúaListaDePuntos **parce pyc**

ContinúaListaDePuntos → **coma id**

```
{ Scene escena = ContinuaListaDePuntos.escena;
  Polygon poligono = ContinuaListaDePuntos.poligono;
  Point p = escena.searchPoint(id.lexema);
  poligono.addPoint(p);
  ContinuaListaDePuntos1.escena = escena;
  ContinuaListaDePuntos1.poligono = poligono;}
```

ContinúaListaDePuntos1

ContinúaListaDePuntos → λ

Ejercicio 5.12

SOLUCIÓN:

Atributos:

- Asig: s (sintetizado) (AssignInst)
- Expr: s (sintetizado) (Set)
- Comp: s (sintetizado) (Set)
- Base: s (sintetizado) (Set)
- Lista: s (sintetizado) (Set)
- Elem: s (sintetizado) (EnumSet)

Acciones semánticas:

Asig → **id igual** *Expr* **pyc**

{ Asig.s = new AsignInst(getVariable(id.lexema), Expr.s); }

Expr → *Comp*

{ Expr.s = Comp.s; }

Expr → *Expr* **union** *Comp*

{ Expr.s = new Union(Expr_1.s, Comp.s); }

Expr → *Expr* **interseccion** *Comp*

{ Expr.s = new Intersection(Expr_1.s, Comp.s); }

Comp → **complemento** *Base*

{ Comp.s = new Complement(Base.s); }

Comp → *Base*

{ Comp.s = Base.s; }

Base → **llave_ab** *Lista* **llave_ce**

{ Base.s = Lista.s; }

Base → **par_ab** *Expr* **par_ce**

{ Base.s = Expr.s; }

Base → **id**

{ Base.s = getVariable(id.lexema); }

Lista → λ

{ Lista.s = new EnumSet(); }

Lista → *Elem*

{ Lista.s = Elem.s; }

Elem → **num**

{ Elem.s = new EnumSet();
Elem.s.addElement(Integer.parseInt(num.lexema)); }

Elem → *Elem* **coma num**

{ Elem.s = Elem_1.s;
Elem.s.addElement(Integer.parseInt(num.lexema));}

Ejercicio 5.13

SOLUCIÓN:

Atributos:

- Asig: s (sintetizado) (AssignInst)
- Expr: s (sintetizado) (Set)
- Comp: s (sintetizado) (Set)
- SigueExpr: h (heredado) (Set), s (sintetizado) (Set)
- Base: s (sintetizado) (Set)
- Lista: s (sintetizado) (Set)
- SigueLista: h (heredado) (EnumSet)
- Elem: s (sintetizado) (EnumSet)

Acciones semánticas:

Asig → **id igual** *Expr* **pyc**
{ Asig.s = new AsignInst(getVariable(id.lexema), Expr.s); }

Expr → *Comp* {SigueExpr.h = Comp.s; } *SigueExpr*
{ Expr.s = SigueExpr.s; }

SigueExpr → **union** *Comp*
{ SigueExpr1.h = new Union(SigueExpr.h, Comp.s); }
SigueExpr1 { SigueExpr.s = SigueExpr1.s; }

SigueExpr → **interseccion** *Comp*
{ SigueExpr1.h = new Intersection(SigueExpr.h, Comp.s); }
SigueExpr1 { SigueExpr.s = SigueExpr1.s; }

SigueExpr → λ { SigueExpr.s = SigueExpr.h; }

Comp → **complemento** *Base*
{ Comp.s = new Complement(Base.s); }

Comp → *Base*
{ Comp.s = Base.s; }

Base → **llave_ab** *Lista* **llave_ce**
{ Base.s = Lista.s; }

Base → **par_ab** *Expr* **par_ce**
{ Base.s = Expr.s; }

Base → **id**
{ Base.s = getVariable(id.lexema); }

Lista → **λ**
{ Lista.s = new EnumSet(); }

Lista → **num** { Lista.s = new EnumSet(); Lista.s.addElement(num);
SigueLista.h = Lista.s; } SigueLista

SigueLista → **λ** {}

SigueLista → **coma num** { SigueLista.h.addElement(num);
SigueLista1.h = SigueLista.h; } *SigueLista1*

Ejercicio 5.14

SOLUCIÓN:

```
AsignInst Asig() :
{
  Token tid;
  Set s;
}
{
  tid = <ID> <IGUAL> s = Expr() <PYC>
  { return new AsignInst(tid.image, s); }
}

Set Expr() :
{
  Set c,s;
  boolean u;
}
{
  s = Comp()
  (
   ( <UNION> { u=true; } | <INTERSECCION> { u=false;} )
    c = Comp()
    { if(u) s = new Union(s,c);
      else s = new Intersection(s,c); }
   )*
  { return s; }
}

Set Comp() :
{
  Set b;
  boolean comp = false;
}
{
  ( <COMPLEMENTO> { comp=true; })?
  b = Base()
  { if(comp) return new Complement(b); else return b; }
}

Set Base() :
{
  Token tid, num;
  Set expr;
  Enum e;
}
{
  (  tid = <ID> { return getVariable(tid.image); }
  | <LLAVEAB> { e = new Enum(); }
    ( num = <NUM> { e.addElement(num.image); }
      ( <COMA> <NUM> { e.addElement(num.image); } )*
    )? <LLAVECE>
    { return e; }
  | <PARAB> expr = Expr() <PARCE> { return expr; }
  )
}
```

Ejercicio 5.15

Ejercicio 5.16

Ejercicio 5.17

Ejercicio 5.18

Ejercicio 5.19

Ejercicio 5.20

Ejercicio 5.21

SOLUCIÓN:

```
Expression Potencia() :
{
   Expression Potencia_s;
   Number num;
   Power last = null;
   Token tk;
}
{
   tk =<NUM>
   {
     num = new Number(Double.parseDouble(tk.image));
     Potencia_s = num;
   }
   (
    <ELEV>
    tk = <NUM>
    {
      num = new Number(Double.parseDouble(tk.image));
      if(last == null) {
        Potencia_s = new Power(Potencia_s, num);
        last = Potencia_s;
      } else {
        last.pow = new Power(last.pow, num);
        last = last.pow;
    }

   )*

   { return Potencia_s; }
}
```

Ejercicio 5.22

Ejercicio 5.23

Ejercicio 5.24

Ejercicio 5.25

Ejercicio 5.26

Ejercicio 5.27

SOLUCIÓN 1:

Esta solución se basa en realizar recorridos sobre las listas de términos, lo que permite representar las listas solo mediante la referencia al primer elemento.

Atributos:

- Expr.s (sintetizado) (Term)
- SigueExpr.s (sintetizado) (Term)
- Factor.s (sintetizado) (Term)

Acciones semánticas:

```
Expr → Factor SigueExpr
        {
          Expr.s = Factor.s;
          Term last = Factor.s;
          while(last.getNext() != null) last = last.getNext();
          last.setNext(SigueExpr.s);
        }
SigueExpr → plus Factor SigueExpr1
        {
          SigueExpr.s = Factor.s;
          Term last = Factor.s;
          while(last.getNext() != null) last = last.getNext();
          last.setNext(SigueExpr1.s);
        }
SigueExpr → minus Factor SigueExpr
        {
          SigueExpr.s = Factor.s;
          Term last = Factor.s;
          last.setFactor( - last.getFactor() );
          while(last.getNext() != null)
              {  last = last.getNext(); last.setFactor( - last.getFactor() );}
          last.setNext(SigueExpr1.s);
        }
```

```
SigueExpr → λ
        {
          SigueExpr.s = null;
        }
Factor → Vector
        {
          Factor.s = new Term( 1.0, Vector.s );
        }
Factor → num prod Factor1
        {
          Factor.s = Factor1.s;
          double f = Double.parseDouble(num.lexema);
          for(Term term = Factor.s; term != null; term = term.getNext() )
              { term.setFactor( f * term.getFactor() ); }
        }
Factor → lpar Expr rpar
        {
          Factor.s = Expr.s;
        }
```

SOLUCIÓN 2:

Una solución más compleja es utilizar referencias al primer y al último elemento de cada lista y utilizar un atributo heredado para propagar el producto del escalar por los vectores. En edste caso habrá que lanzar el símbolo inicial (Expr) considerando como atributo heredado el valor 1.0.

Atributos:

- Expr.first (sintetizado) (Term)
- Expr.last (sintetizado) (Term)
- Expr.h (heredado) (double)
- SigueExpr.first (sintetizado) (Term)
- SigueExpr.last (sintetizado) (Term)

- SigueExpr.h (heredado) (double)
- Factor.first (sintetizado) (Term)
- Factor.last (sintetizado) (Term)
- Factor.h (heredado) (double)

Acciones semánticas:

```
Expr → { Factor.h = Expr.h; } Factor
       { SigueExpr.h = Expr.h; } SigueExpr
       {
          Expr.first = Factor.first;
          Factor.last.setNext( SigueExpr.first) ;
          if( SigueExpr.last == null) Expr.last = Factor.last;
          else Expr.last = SigueExpr.last;
       }
SigueExpr → plus { Factor.h = SigueExpr.h; } Factor
            { SigueExpr1.h = SigueExpr.h; } SigueExpr1
       {
          SigueExpr.first = Factor.first;
          Factor.last.setNext( SigueExpr1.first ) ;
          if( SigueExpr1.last == null) SigueExpr.last = Factor.last;
          else SigueExpr.last = SigueExpr1.last;
       }
SigueExpr → minus { Factor.h = - SigueExpr.h; } Factor
            { SigueExpr1.h = SigueExpr.h; } SigueExpr1
       {
          SigueExpr.first = Factor.first;
          Factor.last.setNext( SigueExpr1.first ) ;
          if( SigueExpr1.last == null) SigueExpr.last = Factor.last;
          else SigueExpr.last = SigueExpr1.last;
       }
```

SigueExpr → λ

{

SigueExpr.first = null;

SigueExpr.last = null;

}

Factor → *Vector*

{

Factor.first = new Term(Factor.h, Vector.s);

Factor.last = Factor.first;

}

Factor → **num** **prod** { $Factor_1$.h = num * Factor.h; } *Factor*$_1$

{

Factor.first = $Factor_1$.first;

Factor.last = $Factor_1$.last;

}

Factor → **lpar** { Expr.h = Factor.h; } *Expr* **rpar**

{

Factor.first = Expr.first;

Factor.last = Expr.last;

}

Ejercicio 5.28

SOLUCIÓN:

Atributos:

- Pedido.s (sintetizado) (Term)
- Lista.first (sintetizado) (Term)
- Lista.last (sintetizado) (Term)
- Lista.h = (heredado) (int)
- SigueLista.last (sintetizado) (Term)
- SigueLista.prev (heredado) (Term)
- SigueLista.h = (heredado) (int)
- Termino.first (sintetizado) (Term)
- Termino.last (sintetizado) (Term)
- Termino.h (heredado) (int)

Acciones semánticas:

Pedido → { Lista.h = 1; } *Lista* {Pedido.s = Lista.first; }

Lista → { Termino.h = Lista.h; } *Término*
{ SigueLista.h = Lista.h, SigueLista.prev = Termino.last; }
SigueLista
{ Lista.first = Termino.first; Lista.last = SigueLista.last; }

SigueLista → **plus** { Termino.h = SigueLista.h; } *Término*
{ SigueLista.prev.setNext(Termino.first);
Termino.first.setPrev(SigueLista.prev);
$SigueLista_1$.h = Lista.h, $SigueLista_1$.prev = Termino.last; }
$SigueLista_1$
{ SigueLista.last = $SigueLista_1$.last; }

SigueLista → λ { SigueLista.last = SigueLista.prev; }

Término → **producto**
{ Termino.first = new Term(Termino.h, producto.lexema);
Termino.last = Termino.first; }

Término → **num** **prod** { $Termino_1$.h = num*Termino.h; } *$Término_1$*
{ Termino.first = $Termino_1$.first; Termino.last = $Termino_1$.last; }

Término → **lpar** { Lista.h = Termino.h; } *Lista* **rpar**
{ Termino.first = Lista.first; Termino.last = Lista.last; }

Ejercicio 5.29

Ejercicio 5.30

SOLUCIÓN:

Atributos:

- Pedido.s (sintetizado) (Term)
- Lista.first (sintetizado) (Term)
- Lista.last (sintetizado) (Term)
- Lista.h = (heredado) (int)
- Termino.first (sintetizado) (Term)
- Termino.last (sintetizado) (Term)
- Termino.h (heredado) (int)

Acciones semánticas:

Pedido → { Lista.h = 1; } *Lista* {Pedido.s = Lista.first; }
Lista → { $Termino_1$.h = Lista.h; }
Término$_1$
{ Lista.first = $Termino_1$.first; Lista.last = $Termino_1$.last; }
(**plus** { $Termino_2$.h = Lista.h; } *Término2*
{ Lista.last.setNext($Termino_2$.first);
$Termino_2$.first.setPrev(Lista.last);
Lista.last = $Termino_2$.last; }
)*
Término → **producto**
{ Termino.first = new Term(Termino.h, producto.lexema);
Termino.last = Termino.first; }
Término → **num** **prod** { $Termino_1$.h = num*Termino.h; } *Término*$_1$
{ Termino.first = $Termino_1$.first; Termino.last = $Termino_1$.last; }
Término → **lpar** { Lista.h = Termino.h; } *Lista* **rpar**
{ Termino.first = Lista.first; Termino.last = Lista.last; }