

Semántica de \mathcal{LTA}

José de Jesús Lavalle Martínez

<http://aleteya.cs.buap.mx/~jlavalle/>

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Licenciatura en Ciencias de la Computación
Fundamentos de Lenguajes de Programación
CCOS 255

Otoño 2020

- 1 Sintaxis de \mathcal{LTA}
- 2 Semántica de \mathcal{LTA}
- 3 Ejercicios
- 4 Asesoría

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, (,)\}$$

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, (,)\}$$

$$d ::= 0|1|\dots|9$$

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, (,)\}$$

$$d ::= 0|1|\dots|9$$

$$n ::= d|nd$$

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, (,)\}$$

$$d ::= 0 | 1 | \dots | 9$$

$$n ::= d | nd$$

$$e ::= n | (e_i + e_d) | (e_i - e_d) | (e_i * e_d)$$

$$d ::= 0|1|\dots|9$$

Implementación de la categoría sintáctica d

$d ::= 0|1|\dots|9$

datatype digi = d0 | d1 | d2 | d3 | d4 |
 d5 | d6 | d7 | d8 | d9 ;

$$d ::= 0|1|\dots|9$$

```
datatype digi = d0 | d1 | d2 | d3 | d4 |  
                d5 | d6 | d7 | d8 | d9 ;
```

Ejemplos:

```
- d0; d4; d9;  
> val it = d0 : digi  
- > val it = d4 : digi  
- > val it = d9 : digi  
-
```

$$n ::= d|nd$$

$$n ::= d | nd$$

```
datatype nume = di of digi | pega of nume * digi;
```

$$n ::= d|nd$$

datatype nume = di **of** digi | pega **of** nume * digi;

Ejemplos: Expresar 5 y 3718 como miembros del tipo nume.

```
- di d5;  
> val it = di d5 : nume  
- pega(pega(pega(di d3, d7), d1), d8);  
> val it = pega(pega(pega(di d3, d7), d1), d8) : nume  
-
```

Implementación de la categoría sintáctica e

$$e ::= n | (e_i + e_d) | (e_i - e_d) | (e_i * e_d)$$

Implementación de la categoría sintáctica e

$$e ::= n | (e_i + e_d) | (e_i - e_d) | (e_i * e_d)$$

```
datatype exar = nu of nume |  
                ma of exar * exar |  
                me of exar * exar |  
                mu of exar * exar ;
```

Implementación de la categoría sintáctica e

$$e ::= n | (e_i + e_d) | (e_i - e_d) | (e_i * e_d)$$

```
datatype exar = nu of nume |  
                ma of exar * exar |  
                me of exar * exar |  
                mu of exar * exar ;
```

Ejemplos: Expresar $(10 - (5 * 2))$ como miembro del tipo `exar`.

```
- pega(di d1, d0); di d5; di d2;  
> val it = pega(di d1, d0) : nume  
- > val it = di d5 : nume  
- > val it = di d2 : nume
```

Implementación de la categoría sintáctica e

$$e ::= n | (e_i + e_d) | (e_i - e_d) | (e_i * e_d)$$

```
datatype exar = nu of nume |  
                ma of exar * exar |  
                me of exar * exar |  
                mu of exar * exar ;
```

Ejemplos: Expresar $(10 - (5 * 2))$ como miembro del tipo `exar`.

```
- pega(di d1, d0); di d5; di d2;  
> val it = pega(di d1, d0) : nume  
- > val it = di d5 : nume  
- > val it = di d2 : nume  
  
- mu(nu(di d5), nu(di d2));  
> val it = mu(nu(di d5), nu(di d2)) : exar  
- nu(pega(di d1, d0));  
> val it = nu(pega(di d1, d0)) : exar
```


Implementación de la categoría sintáctica e

$$e ::= n | (e_i + e_d) | (e_i - e_d) | (e_i * e_d)$$

```
datatype exar = nu of nume |  
                ma of exar * exar |  
                me of exar * exar |  
                mu of exar * exar ;
```

Ejemplos: Expresar $(10 - (5 * 2))$ como miembro del tipo `exar`.

```
- pega(di d1, d0); di d5; di d2;  
> val it = pega(di d1, d0) : nume  
- > val it = di d5 : nume  
- > val it = di d2 : nume  
  
- mu(nu(di d5), nu(di d2));  
> val it = mu(nu(di d5), nu(di d2)) : exar  
- nu(pega(di d1, d0));  
> val it = nu(pega(di d1, d0)) : exar  
  
- me(nu(pega(di d1, d0)), mu(nu(di d5), nu(di d2)));  
> val it = me(nu(pega(di d1, d0)), mu(nu(di d5), nu(di d2))) : exar  
-
```

$$\llbracket \cdot \rrbracket : \mathcal{LTA} \rightarrow \mathbb{Z}$$

$$\llbracket \cdot \rrbracket : \mathcal{LTA} \rightarrow \mathbb{Z}$$

$$\llbracket 0 \rrbracket = 0, \dots, \llbracket 9 \rrbracket = 9$$

$$\llbracket \cdot \rrbracket : \mathcal{LTA} \rightarrow \mathbb{Z}$$

$$\llbracket 0 \rrbracket = 0, \dots, \llbracket 9 \rrbracket = 9$$

$$\llbracket nd \rrbracket = ((\llbracket n \rrbracket * 10) + \llbracket d \rrbracket)$$

$$\llbracket \cdot \rrbracket : \mathcal{LTA} \rightarrow \mathbb{Z}$$

$$\llbracket 0 \rrbracket = 0, \dots, \llbracket 9 \rrbracket = 9$$

$$\llbracket nd \rrbracket = ((\llbracket n \rrbracket * 10) + \llbracket d \rrbracket)$$

$$\llbracket (e_i + e_d) \rrbracket = (\llbracket e_i \rrbracket + \llbracket e_d \rrbracket)$$

$$\llbracket \cdot \rrbracket : \mathcal{LTA} \rightarrow \mathbb{Z}$$

$$\llbracket 0 \rrbracket = 0, \dots, \llbracket 9 \rrbracket = 9$$

$$\llbracket nd \rrbracket = ((\llbracket n \rrbracket * 10) + \llbracket d \rrbracket)$$

$$\llbracket (e_i + e_d) \rrbracket = (\llbracket e_i \rrbracket + \llbracket e_d \rrbracket)$$

$$\llbracket (e_i - e_d) \rrbracket = (\llbracket e_i \rrbracket - \llbracket e_d \rrbracket)$$

$$\llbracket \cdot \rrbracket : \mathcal{LTA} \rightarrow \mathbb{Z}$$

$$\llbracket 0 \rrbracket = 0, \dots, \llbracket 9 \rrbracket = 9$$

$$\llbracket nd \rrbracket = ((\llbracket n \rrbracket * 10) + \llbracket d \rrbracket)$$

$$\llbracket (e_i + e_d) \rrbracket = (\llbracket e_i \rrbracket + \llbracket e_d \rrbracket)$$

$$\llbracket (e_i - e_d) \rrbracket = (\llbracket e_i \rrbracket - \llbracket e_d \rrbracket)$$

$$\llbracket (e_i * e_d) \rrbracket = (\llbracket e_i \rrbracket * \llbracket e_d \rrbracket)$$

Implementación de la semántica de d

```
datatype digi = d0 | d1 | d2 | d3 | d4 |  
                d5 | d6 | d7 | d8 | d9 ;
```


Implementación de la semántica de d

```
datatype digi = d0 | d1 | d2 | d3 | d4 |  
                d5 | d6 | d7 | d8 | d9 ;
```

$\llbracket 0 \rrbracket = 0, \dots, \llbracket 9 \rrbracket = 9$

Implementación de la semántica de d

```
datatype digi = d0 | d1 | d2 | d3 | d4 |  
                d5 | d6 | d7 | d8 | d9;
```

$\llbracket 0 \rrbracket = 0, \dots, \llbracket 9 \rrbracket = 9$

```
fun digi2int d0 = 0  
    | digi2int d1 = 1  
    | digi2int d2 = 2  
    | digi2int d3 = 3  
    | digi2int d4 = 4  
    | digi2int d5 = 5  
    | digi2int d6 = 6  
    | digi2int d7 = 7  
    | digi2int d8 = 8  
    | digi2int d9 = 9;
```

```
datatype nume = di of digi | pega of nume * digi;
```

datatype nume = di **of** digi | pega **of** nume * digi;

$$\llbracket nd \rrbracket = ((\llbracket n \rrbracket * 10) + \llbracket d \rrbracket)$$

```
datatype nume = di of digi | pega of nume * digi;
```

$$\llbracket nd \rrbracket = ((\llbracket n \rrbracket * 10) + \llbracket d \rrbracket)$$

```
fun nume2int(di(any)) =  
| nume2int(pega(n,d)) =
```

Implementación de la semántica de e

```
datatype exar = nu of nume |  
                ma of exar * exar |  
                me of exar * exar |  
                mu of exar * exar ;
```

Implementación de la semántica de e

```
datatype exar = nu of nume |  
                ma of exar * exar |  
                me of exar * exar |  
                mu of exar * exar ;
```

$$\llbracket e_i + e_d \rrbracket = (\llbracket e_i \rrbracket + \llbracket e_d \rrbracket)$$

$$\llbracket e_i - e_d \rrbracket = (\llbracket e_i \rrbracket - \llbracket e_d \rrbracket)$$

$$\llbracket e_i * e_d \rrbracket = (\llbracket e_i \rrbracket * \llbracket e_d \rrbracket)$$

Implementación de la semántica de e

```
datatype exar = nu of nume |  
                ma of exar * exar |  
                me of exar * exar |  
                mu of exar * exar ;
```

$$\llbracket e_i + e_d \rrbracket = (\llbracket e_i \rrbracket + \llbracket e_d \rrbracket)$$

$$\llbracket e_i - e_d \rrbracket = (\llbracket e_i \rrbracket - \llbracket e_d \rrbracket)$$

$$\llbracket e_i * e_d \rrbracket = (\llbracket e_i \rrbracket * \llbracket e_d \rrbracket)$$

```
fun exar2int(nu(any)) =  
| exar2int(ma(ei, ed)) =  
| exar2int(me(ei, ed)) =  
| exar2int(mu(ei, ed)) =
```


Para convertir un entero en nume

$$3718 \div 10 = 371$$

$$371 \div 10 = 37$$

$$37 \div 10 = 3$$

3

$$3718 \bmod 10 = 8$$

$$371 \bmod 10 = 1$$

$$37 \bmod 10 = 7$$

```
pega(pega(pega(di d3, d7), d1), d8));
```

Para convertir un entero en nume

$$3718 \div 10 = 371$$

$$371 \div 10 = 37$$

$$37 \div 10 = 3$$

3

$$3718 \bmod 10 = 8$$

$$371 \bmod 10 = 1$$

$$37 \bmod 10 = 7$$

```
pega(pega(pega(di d3, d7), d1), d8));
```

Deben implementar las funciones:

```
> val int2digi = fn : int -> digi
```

```
> val int2nume = fn : int -> nume
```

```
-
```

Para convertir un entero en nume

$$3718 \div 10 = 371$$

$$371 \div 10 = 37$$

$$37 \div 10 = 3$$

3

$$3718 \bmod 10 = 8$$

$$371 \bmod 10 = 1$$

$$37 \bmod 10 = 7$$

```
pega(pega(pega(di d3, d7), d1), d8));
```

Deben implementar las funciones:

```
> val int2digi = fn : int -> digi
> val int2nume = fn : int -> nume
-
```

En ML el cociente de dos enteros es un operador infijo llamado `div` y el residuo también es un operador infijo y se llama `mod`.

Para entregar sus reportes

- Debe escribir el código usando:

```
\begin{lstlisting}[language=ML]
```

```
\end{lstlisting}
```

- La respuesta del intérprete la reportan usando:

```
\begin{verbatim}
```

```
\end{verbatim}
```

- No debe cambiar los nombres de los tipos, ni de los constructores, ni de las funciones.

- Deben incluir una captura de pantalla de la interacción que hacen con el intérprete.

Para insertar una imagen en \LaTeX


Deben incluir en el preámbulo el paquete `graphicx` mediante `\usepackage{graphicx}`.

```
\begin{center}  
\includegraphics[scale=.225]{ejemploCaptura}  
\end{center}
```


Para insertar una imagen en \LaTeX

Deben incluir en el preámbulo el paquete `graphicx` mediante `\usepackage{graphicx}`.

```
\begin{center}  
\includegraphics[scale=.225]{ejemploCaptura}  
\end{center}
```



```
datatype  
exar = nu of nume | ma of exar * exar | me of exar * exar | mu of exar * exar;> New type names: =digi  
datatype digi =  
(digi,  
 {con d0 : digi,  
   con d1 : digi,  
   con d2 : digi,  
   con d3 : digi,  
   con d4 : digi,  
   con d5 : digi,  
   con d6 : digi,  
   con d7 : digi,  
   con d8 : digi,  
   con d9 : digi})  
con d0 = d0 : digi  
con d1 = d1 : digi  
con d2 = d2 : digi  
con d3 = d3 : digi  
con d4 = d4 : digi  
con d5 = d5 : digi  
con d6 = d6 : digi  
con d7 = d7 : digi  
con d8 = d8 : digi  
con d9 = d9 : digi  
-> New type names: =nume  
datatype nume =  
(nume, {con di : digi -> nume, con pega : nume * digi -> nume})  
con di = fn : digi -> nume  
con pega = fn : nume * digi -> nume
```

- 1 Implemente las siguientes funciones:
 - 1 `int2digi = fn : int -> digi`
 - 2 `int2nume = fn : int -> nume`
 - 3 Recuerde que `digi2int = fn : digi -> int` se implementó en esta presentación.
 - 4 `nume2int = fn : nume -> int`
 - 5 `exar2int = fn : exar -> int`
- 2 Pruebe su implementación con los siguientes casos de prueba, incluya tres de su elección:
 - 1 `exar2int(me(ma(nu(int2nume(17)),nu(int2nume(15))),nu(int2nume(8))));`
 - 2 `exar2int(me(ma(nu(int2nume(3000)),nu(int2nume(2000))),nu(int2nume(2000))));`

```
fun suma(0, n) = n
| suma(m, n) = 1 + suma(m-1, n);
fun sumales(z, n) = n
| sumales(s(m), n) = s(sumales(m, n));
```