Evaluación semanal 3

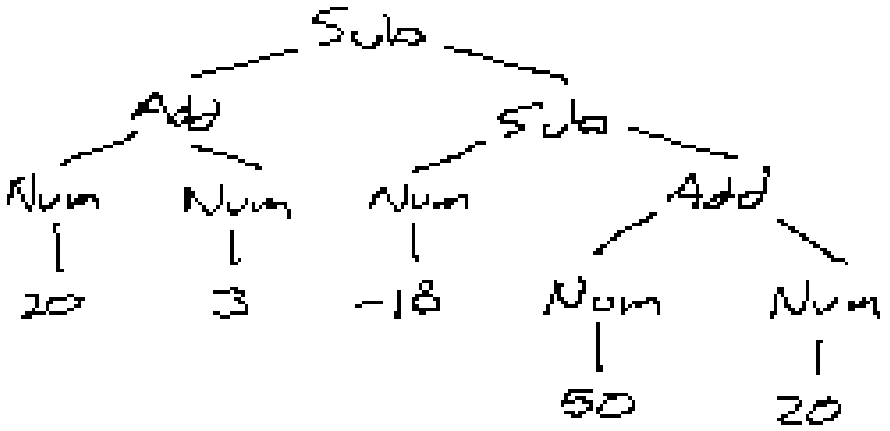
Santiago González Tamariz
Lenguajes de Programación

1. Dadas las siguientes expresiones en sintaxis concreta de nuestro lenguaje **MiniLisp**

   - Obtener su sintaxis abstracta
   - Evaluarlas usando las reglas de semántica natural
   - Evaluarlas usando las reglas de semántica estructural

   (a) `(- (+ 20 3) (- -18 (+ 50 20)))`

   Sintaxis abstracta



   `Sub( Add( Num(20) Num(3) ) Sub( Num(-18) Add( Num(50) Num(20) ) ) )`

   Semántica natural

$$\dfrac{\dfrac{\dfrac{}{\texttt{Num(20)} \Rightarrow \texttt{Num(20)}} \quad \dfrac{}{\texttt{Num(3)} \Rightarrow \texttt{Num(3)}}}{\texttt{Add(Num(20) Num(3))} \Rightarrow \texttt{Num(23)}} \quad \dfrac{\dfrac{}{\texttt{Num(-18)} \Rightarrow \texttt{Num(-18)}} \quad \dfrac{\dfrac{}{\texttt{Num(50)} \Rightarrow \texttt{Num(50)}} \quad \dfrac{}{\texttt{Num(20)} \Rightarrow \texttt{Num(20)}}}{\texttt{Add(Num(50) Num(20))} \Rightarrow \texttt{Num(70)}}}{\texttt{Sub(Num(-18) Add(Num(50) Num(20)))} \Rightarrow \texttt{Num(-88)}}}{\texttt{Sub(Add(Num(20) Num(3)) Sub(Num(-18) Add(Num(50) Num(20))))} \Rightarrow \texttt{Num(111)}}$$

   Semántica estructural

   `Sub( Add( Num(20) Num(3) ) Sub( Num(-18) Add( Num(50) Num(20) ) ) )` $\rightarrow$
   `Sub( Num(23) Sub( Num(-18) Add( Num(50) Num(20) ) ) )` $\rightarrow$
   `Sub( Num(23) Sub( Num(-18) Num(70) ) )` $\rightarrow$
   `Sub( Num(23) Num(-88) )` $\rightarrow$
   `Num(111)` $\rightarrow$ `Num(111)`

   (b) `(not (+ 1 (- 3 (+ -8 1))))`

   (c) `(not (not (+ 3 5)))`

2. Extender la batería de operaciones de **MiniLisp**
   En los tres casos, deberás usar la notación formal que vimos en clase

   - Dar la gramática libre de contexto modificada (en notación EBNF) añadiendo las nuevas construcciones del lenguaje
   - Modificar las reglas de sintaxis abstracta para considerar los nuevos constructores
   - Extender las reglas de semántica natural y estructural

   (a) Especificar un nuevo constructor `*` para la multiplicación binaria de expresiones aritméticas. Por ejemplo

   ```
   > (* 20 2)
   40
   ```

(b) Especificar un nuevo constructor `/` para la división binaria de expresiones aritméticas. Consideren que no se pueden realizar divisiones entre cero. Por ejemplo:

```
> (/ 20 2)
10
> (/ 10 0)
error: División entre cero
```

(c) Especificar un nuevo constructor `add1` que dada una expresión, incrementa en uno su valor. Por ejemplo:

```
> (add1 10)
11
```

(d) Especificar un nuevo constructor `sub1` que dada una expresión, decrementa en uno su valor. Por ejemplo:

```
> (sub1 10)
9
```

(e) Especificar un nuevo constructor `sqrt` que dada una expresión, obtiene la raíz cuadrada de dicha expresión. Consideren que no se pueden calcular raíces cuadradas de números negativos. Por ejemplo:

```
> (sqrt 81)
9
> (sqrt -2)
error: Raíz negativa
```