Universidad Nacional Autónoma de México

Facultad de Ciencias

Lenguajes de Programación | 7098

Semanal 3: Sintaxis a., Semánticas natural y estructurada |

Sosa Romo Juan Mario | 320051926

Legorreta Esparragoza Juan Luis | 319317532

Gómez López Erik Eduardo | 320258211

27/08/24

1. **Dadas las siguientes expresiones en sintaxis concreta de nuestro lenguaje MiniLisp (a) obtener su sintaxis abstracta, (b) evaluarlas usando las reglas de semántica natural y (c) evaluarlas usando las reglas de semántica estructural. Todas las reglas las podrán consultar en la Nota de Clase 6 y la Nota de Clase 7.**

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

   i. **Obtener su sintaxis abstracta**

   $$Sub(Add(Num(20), Num(3)), (Sub(Num(-18), (Add(Num(50), Num(20))))))$$

   ii. **Evaluarla usando las reglas de semántica natural (paso grande).**

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

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

   iii. **Evaluarla usando las reglas de semántica estructural (paso pequeño).**

$$\begin{aligned} Sub(Add(Num(20), Num(3)), (Sub(Num(-18), (Add(Num(50), Num(20))))))) &\rightarrow \\ Sub(Num(23), (Sub(Num(-18), (Add(Num(50), Num(20))))))) &\rightarrow \\ Sub(Num(23), (Sub(Num(-18), (Num(70)))))) &\rightarrow \\ Sub(Num(23), (-88)) &\rightarrow \\ Num(111) & \end{aligned}$$

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

   i. **Obtener su sintaxis abstracta**

   $$Not(Add(Num(1), (Sub(Num(3), (Add(Num(-8), Num(1))))))))$$

   ii. **Evaluarla usando las reglas de semántica natural (paso grande).**

$$\dfrac{\dfrac{}{Num(1) \Rightarrow Num(1)} \quad \dfrac{\dfrac{}{Num(3) \Rightarrow Num(3)} \quad \dfrac{\dfrac{}{Num(-8) \Rightarrow Num(-8)} \quad \dfrac{}{Num(1) \Rightarrow Num(1)}}{Add(Num(-8), Num(1)) \Rightarrow Num(-7)}}{Sub(Num(3), (Add(Num(-8), Num(1))) \Rightarrow Num(-10)}}{Add(Num(1), (Sub(Num(3), (Add(Num(-8), Num(1)))))) \Rightarrow Num(-9)}$$

$$\dfrac{Add(Num(1), (Sub(Num(3), (Add(Num(-8), Num(1)))))) \Rightarrow Num(-9)}{Not(Add(Num(1), (Sub(Num(3), (Add(Num(-8), Num(1))))))) \Rightarrow Boolean(False)}$$

iii. **Evaluarla usando las reglas de semántica estructural (paso pequeño).**

$$Not(Add(Num(1), (Sub(Num(3), (Add(Num(-8), Num(1))))))) \rightarrow$$
$$Not(Add(Num(1), (Sub(Num(3), Num(-7))) \rightarrow$$
$$Not(Add(Num(1), (Num(-10)) \rightarrow$$
$$Not(Num(-9)) \rightarrow$$
$$Boolean(False)$$

(c) **Sintáxis abstracta**

$$ASA : Not(Not(Add(Num(3)Num(5))))$$

Evaluación de semática natural

$$\frac{\dfrac{\dfrac{Num(3) \Rightarrow Num(3) \qquad Num(5) \Rightarrow Num(5)}{Add(Num(3), Num(5)) \Rightarrow Num(8)}}{Not(Add(Num(3), Num(5))) \Rightarrow Boolean(False)}}{Not(Not(Add(Num(3), Num(5)))) \Rightarrow Boolean(True)}$$

$$(not(not(+ \quad 3 \quad 5)))$$
$$not(not(Add(Num(3)\ Num(5))))$$
$$\rightarrow not(not(Num(8)))$$
$$\rightarrow not(True)$$
$$\rightarrow False$$

2. **Como segundo ejercicio deberán extender la batería de operaciones de MiniLisp, para ello deberán (a) dar la gramática libre de contexto modificada (en notación EBNF) añadiendo las nuevas construcciones del lenguaje, (b) modificar las reglas de sintaxis abstracta para considerar los nuevos constructores y finalmente (c) extender las reglas de semántica natural y estructural. En los tres casos, deberás usar la notación formal que vimos en clase.**

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

   - **Dar la gramática libre de contexto modificada (en notación EBNF) añadiendo las nuevas construcciones del lenguaje.**

     Básicamente vamos a usar las mismas que ya tenemos hasta este punto modificando 'Ex-

pression':

$$
\begin{aligned}
&\ldots \\
< Expr > ::= &< Int > \\
&\mid (+ \ < Expr > < Expr >) \\
&\mid (- \ < Expr > < Expr >) \\
&\mid (* \ < Expr > < Expr >) \\
&\ldots
\end{aligned}
$$

- **Modificar las reglas de sintaxis abstracta para considerar los nuevos constructores**

$$\frac{i \ ASA \qquad d \ ASA}{Mul(i,d) \ ASA}$$

- **Extender las reglas de ambas semánticas**

  - **Semántica Natural:**

$$\frac{i \Rightarrow Num(n_1) \qquad d \Rightarrow Num(n_2)}{Mul(i,d) \Rightarrow Num(n_1 * n_2)}$$

  - **Semántica Estructural**
    Se dividen en 3 casos:

    * **Caso 1:** Ninguno esta reducido, reducimos el izquierdo:

$$\frac{i \rightarrow i'}{Mul(i,d) \rightarrow Mul(i',d)}$$

    * **Caso 2:** El argumento de la izquierda es un 'Num' pero el de la derecha no, lo reducimos:

$$\frac{d \rightarrow d'}{Mul(Num(n_1),d) \rightarrow Mul(Num(n_1),d')}$$

    * **Caso 3:** Ambos argumentos son 'Num' así que hacemos la operación tal cual.:

$$\frac{}{Mul(Num(n_1),Num(n_2)) \rightarrow Num(n_1 * n_2)}$$

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

- **Dar la gramática libre de contexto modificada (en notación EBNF) añadiendo las nuevas construcciones del lenguaje.**

Nuevamente vamos a usar las mismas que ya tenemos hasta este punto modificando 'Expression':

$$
\begin{aligned}
&\ldots \\
< Expr > ::= &< Int > \\
&\mid (+ \ < Expr > < Expr >) \\
&\mid (- \ < Expr > < Expr >) \\
&\mid (* \ < Expr > < Expr >) \\
&\mid (/ \ < Expr > < Expr >) \\
&\ldots
\end{aligned}
$$

- **Modificar las reglas de sintaxis abstracta para considerar los nuevos constructores**

$$\frac{i \ ASA \qquad d \ ASA}{Div(i,d) \ ASA}$$

- **Extender las reglas de ambas semánticas**

  - **Semántica Natural:**

$$\frac{i \Rightarrow Num(n_1) \qquad d \Rightarrow Num(n_2)}{Div(i,d) \Rightarrow Num(n_1/n_2)}$$

    Si el divisor **d** es cero:

$$\frac{i \Rightarrow Num(n_1) \qquad d \Rightarrow Num(n_2)}{Div(i,d) \Rightarrow ERROR \quad n_2 = 0}$$

  - **Semántica Estructural**
    Se dividen en 3 casos:

    * **Caso 1:** Ninguno esta reducido, reducimos el izquierdo:

$$\frac{i \rightarrow i'}{Div(i,d) \rightarrow Div(i',d)}$$

    * **Caso 2:** El argumento de la izquierda es un 'Num' pero el de la derecha no, lo reducimos:

$$\frac{d \rightarrow d'}{Div(Num(n_1),d) \rightarrow Div(Num(n_1),d')}$$

    * **Caso 3:** Ambos argumentos son 'Num' así que hacemos la operación tal cual.:

$$\frac{}{Div(Num(n_1),Num(n_2)) \rightarrow Div(n_1/n_2)}$$

      Si el divisor **d** es cero:

$$\frac{}{Div(Num(n_1), Num(n_2)) \rightarrow ERROR \quad n_2 = 0}$$

(c)

$$\begin{array}{l} \ldots \\ < Expr > ::= < Int > \\ \qquad\qquad | (+ \ < Expr > < Expr >) \\ \qquad\qquad | (- \ < Expr > < Expr >) \\ \qquad\qquad | (* \ < Expr > < Expr >) \\ \qquad\qquad | (add1 \ < Expr >) \\ \ldots \end{array}$$

$$\frac{i \qquad ASA}{Add1(i) \ ASA}$$

En semántica natural

$$\frac{i \Rightarrow Num(i')}{Add1(i) \Rightarrow Num(i'+1)}$$

$$\frac{i \Rightarrow \ Bool(i')}{ERROR}$$

(d) **Especificar un nuevo constructor 'sub1' que dada una expresión, decrementa en uno su valor.**

- **Dar la gramática libre de contexto modificada (en notación EBNF) añadiendo las nuevas construcciones del lenguaje.**

  Básicamente vamos a usar las mismas que ya tenemos hasta este punto modificando 'Expression':

$$\begin{array}{l} \ldots \\ < Expr > ::= < Int > \\ \qquad\qquad | (+ \ < Expr > < Expr >) \\ \qquad\qquad | (- \ < Expr > < Expr >) \\ \ldots \\ \qquad\qquad | (sub1 \ < Expr >) \\ \ldots \end{array}$$

- **Modificar las reglas de sintaxis abstracta para considerar los nuevos constructores**

$$\frac{i \ ASA}{sub1(i) \ ASA}$$

- **Extender las reglas de ambas semánticas**

  - **Semántica Natural:**

$$\frac{i \Rightarrow Num(n_1)}{sub1(i) \Rightarrow Num(n_1 - 1)}$$

  - **Semántica Estructural**
    Se dividen en 2 casos:

    * **Caso 1:** El argumento ya es un 'Num', hacemos la operación:

$$\frac{}{sub1(Num(i)) \rightarrow Num(i-1)}$$

    * **Caso 2:** El argumento no es un 'Num' entonces lo reducimos:

$$\frac{i \rightarrow i'}{sub1(i) \rightarrow sub1(i')}$$

(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:**

- **Dar la gramática libre de contexto modificada (en notación EBNF) añadiendo las nuevas construcciones del lenguaje.**

  Nuevamente vamos a usar las mismas que ya tenemos hasta este punto modificando 'Expression':

$$\begin{aligned}
&\ldots \\
< Expr > ::= &< Int > \\
&\mid (+ \ < Expr > < Expr >) \\
&\mid (- \ < Expr > < Expr >) \\
&\mid (* \ < Expr > < Expr >) \\
&\mid (/ \ < Expr > < Expr >) \\
&\ldots \\
&\mid (sqrt \ < Expr >) \\
&\ldots
\end{aligned}$$

- **Modificar las reglas de sintaxis abstracta para considerar los nuevos constructores**

$$\frac{i \ ASA}{Sqrt(i) \ ASA}$$

- **Extender las reglas de ambas semánticas**

  - **Semántica Natural:**

$$\frac{i \Rightarrow Num(n)}{Sqrt(i) \Rightarrow Num(\sqrt{n})}$$

    Si la expresión es un número negativo:

$$\frac{i \Rightarrow Num(n) \quad n < 0}{Sqrt(i) \Rightarrow ERROR}$$

  - **Semántica Estructural**
    Se dividen en 3 casos:

    * **Caso 1:** El argumento no está reducido:

$$\frac{i \rightarrow i'}{Sqrt(i) \rightarrow Sqrt(i')}$$

    * **Caso 2:** El argumento ya es un número positivo o cero:

$$\frac{}{Sqrt(Num(n)) \rightarrow Num(\sqrt{n})}$$

    * **Caso 3:** El argumento es un número negativo:

$$\frac{}{Sqrt(Num(n)) \rightarrow ERROR \quad n < 0}$$