



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Lenguajes de Programacion Examen Parcial I



■ Edgar Montiel Ledesma
317317794

■ Carlos Daniel Cortes Jimenez
420004846

1. Chon Hacker quiere desarrollar un lenguaje calculador para expresiones aritméticas en notación polaca. La notación polaca es un antiguo sistema que no requiere de paréntesis, para lograrlo mueve los operadores al final de la expresión, es decir, utiliza notación posfija. Por ejemplo, la expresión $1 + 2$ se convierte en $21+$ y la expresión $1 - (3 + 2)$ corresponde a $132 + -$. Este lenguaje evalúa las expresiones metiendo cada símbolo a una pila hasta que se encuentra un operador, al ver un operador saca los dos símbolos que se encuentran en el tope de la pila, evalúa la operación con ellos y agrega el resultado a la pila. Una gramática con la que se puede definir este lenguaje es la siguiente:

$\text{sym} ::= n \mid + \mid - \mid *$

$\text{rpn} ::= \epsilon \mid \text{sym rpn}$

Responde a los siguientes incisos:

- a) Con la gramática anterior se pueden construir programas que no pertenecen al lenguaje como $+12$ o $1 + 2$. Para tratar con esto Chon Hacker te pide definir mediante reglas de inferencia una semántica estática que verifique que la expresión es correcta según la definición de la notación polaca. Para esto se define el juicio e **correct** que indica que la expresión e está correctamente en notación polaca.

Definimos el juicio e **correct** que indica que la expresión e está correctamente en notación polaca, en seguida tenemos nuestras reglas de inferencia:

- 1.- Si e es un número, entonces:

$$\frac{}{n \text{ correct}}$$

- 2.- Si e es de la forma $e_1 e_2 \text{ op}$, donde e_1 **correct** y e_2 **correct**, y op es un operador $(+, -, *)$, entonces:

$$\frac{e_1 \text{ correct} \quad e_2 \text{ correct}}{+ \text{ correct}} \quad \frac{e_1 \text{ correct} \quad e_2 \text{ correct}}{* \text{ correct}} \quad \frac{e_1 \text{ correct} \quad e_2 \text{ correct}}{- \text{ correct}}$$

- 3.- Si $e = \epsilon$ (cadena vacía), entonces e es correcta

$$\overline{\epsilon \text{ correct}}$$

b) Usando la definición del inciso anterior verifica que $1243 + * -$ correct

- 1 correct (según la Regla 1).
- 2 correct (según la Regla 1).
- 4 correct (según la Regla 1).
- 3 correct (según la Regla 1).
- 4 3 + correct (según la Regla 2, ya que 4 y 3 son correctos).
- 2 4 3 + * correct (según la Regla 2, ya que 2 y 4 3 + son correctos).
- 1 2 4 3 + * - correct (según la Regla 2, ya que 1 y 2 4 3 + * son correctos).

Por lo tanto la expresión "1 2 4 3 + * -" cumple con todas las reglas de inferencia y, entonces, es considerada correcta según la definición del juicio "e correct".

c) Define una semántica operacional de paso grande para este lenguaje calculador mediante la relación \Downarrow que evalúa los programas del lenguaje. Puede ser de ayuda leer el programa de derecha a izquierda.

Nuestra semantica operacional de paso grande es la sigueinte:

1.- Regla para números: Un número se evalúa a sí mismo

$$\overline{n \Downarrow n}$$

2.- Regla para operadores (op) y expresiones (sym rpn) con operadores:

Si tenemos una expresión que es una operación con dos operandos (números) y un operador, entonces se evalúa de la siguiente forma:

$$\frac{sym \ rpn \Downarrow n_2 \quad sym \Downarrow n_1 \quad op(sym, n_1, n_2) = n}{sym \ rpn \Downarrow n}$$

Donde:

- *sym* es el operador
- *rpn* es el resto de la expresión
- n_1 es el resultado de evaluar el primer operando
- n_2 es el resultado de evaluar el segundo operando
- $op(sym, n_1, n_2)$ es una función que toma el operador *sym*, el primer operando n_1 y el segundo operando n_2 y devuelve el resultado de la operación.

3.- Regla para espresiones vacía(ϵ):Una expresión vacía no tiene valor

$$\overline{\epsilon \Downarrow error}$$

- d) Define una semántica operacional de paso pequeño con estados de la forma $s \models e$ en donde s es la pila que almacena los símbolos (la pila vacía se denota como \circ) y e es la expresión del lenguaje. Para esto indica claramente cuáles son los estados iniciales y finales y define la relación de transición \rightarrow_{rpn} que modela la evaluación del programa.

Estados iniciales y finales:

- Los estados iniciales se representan cuando la pila s está vacía (\circ) y la expresión e es la expresión completa que se desea evaluar. Inicialmente, la pila está vacía.
- Los estados finales se representan cuando la pila s contiene un solo valor numérico y la expresión e está vacía (ϵ), lo que significa que la evaluación ha concluido y el valor resultante se encuentra en la cima de la pila.

Ahora, definamos las reglas de transición para la relación \rightarrow_{rpn} .

1.- Regla para números:

$$s \models n \rightarrow_{rpn} s \models n$$

Cuando el siguiente símbolo en la expresión es un número, se coloca en la pila de símbolos y se quita de la expresión.

2.- Regla para operadores (+, -, *):

$$n_1 \ n_2 \ s \models op \ e \rightarrow_{rpn} (n_1 \ op \ n_2) \ s \models e$$

Cuando el siguiente símbolo en la expresión es un operador, se toman los dos números superiores en la pila y se realiza la operación op con ellos. El resultado se coloca de nuevo en la pila, y el operador se quita de la expresión.

3.- Regla para expresiones vacías(ϵ):

$$n \ s \models \epsilon \rightarrow_{rpn} n \ s \models \epsilon$$

Cuando la expresión está vacía, la evaluación ha terminado. El resultado es el número nn , y la expresión sigue vacía.

- e) Prueba que las semánticas definidas en los incisos anteriores son equivalentes, para esto:

- Demuestra que para cualquier expresión correcta e del lenguaje, si $e \Downarrow v$ entonces $\circ \models e \rightarrow_{rpn}^* \circ \models v$.
- Prueba que para cualquier expresión correcta e del lenguaje $\circ \models e \rightarrow_{rpn}^* \circ \models v$ de tal forma que $\circ \models v$ es final, entonces $e \Downarrow v$.

1.-Vamos a demostrar por induccion en la evaluación de paso grande:

Caso Base: Para una expresión atómica n si $e = n$, entonces en la semantica de paso pequeño, $\circ \models n \rightarrow_{rpn} \circ \models n$ (por la regla de número). Esto es consistente con la evaluación de paso grande $n \Downarrow n$

Hipótesis de Inducción(H.I.): Supongamos que para cualquier expresión e_1 tal que $e_1 \Downarrow v_1$, se cumple que $\circ \models e_1 \rightarrow_{rpn}^* \circ \models v_1$.

Paso Inductivo: Consideremos una expresión e de la forma $sym\ e_1\ rpn$, donde $e_1 \Downarrow v_1$ y sym es un operador. Por la evaluación de paso grande, tenemos $e \Downarrow v$, y por la regla de operador, $v = op(sym, v_1, v_2)$, donde v_2 es el resultado de evaluar rpn .

Por **H.I.** $\circ \models e_1 \rightarrow_{rpn}^* \circ \models v_1$, y dado que las reglas de paso pequeño son deterministas, la evaluación de rpn desde $\circ \models e_1$ tambien conduce a $\circ \models v_1$. Entonces, tenemos $\circ \models sym\ e_1\ rpn \rightarrow_{rpn}^* \circ \models op(sym, v_1, v_2) = \circ \models v$.

Por lo tanto, se demostró que si $e \Downarrow v$ entonces $\circ \models e \rightarrow_{rpn}^* \circ \models v$.

2.-Vamos a demostrar por inducción en el número de pasos en la evaluación de paso pequeño $\circ \models e \rightarrow_{rpn}^* \circ \models v$:

Caso Base: Si la evaluación se completa en un solo paso, significa que e es una expresión atómica n y $v = n$ Esto es consistente con la evaluación de paso grande $n \Downarrow n$.

Hipótesis de Inducción(H.I.): Supongamos que después de k pasos, $\circ \models e \rightarrow_{rpn}^* \circ \models v$ y $\circ \models v$ es un estado final.

Paso Inductivo: Consideremos el paso $k + 1$ en a evaluación. Tenemos $\circ \models e \rightarrow_{rpn}^* \circ \models v$ y $\circ \models v$ en $k + 1$ pasos, lo que implica que $\circ \models e \rightarrow_{rpn}^* \circ' \models v'$ en k pasos, donde v' es el estado después del paso $k + 1$. Dado que $\circ \models v$ es final, esto significa que no hay más pasos para v' , lo que implica que $v = v'$. Por **H.I.**, si $\circ' \models v' = \circ \models v$, entonces $e \Downarrow v$.

Por lo tanto, se demostró que si $\circ \models e \rightarrow_{rpn}^* \circ \models v$ y $\circ \models v$ es final, entonces $e \Downarrow v$.

Entonces podemos decir que las semánticas definidas en los incisos anteriores son **equivalentes**.

2. Lo numerales de *Barendregt*, denotados \widehat{n} , se definen como sigue:

$$\begin{aligned} \widehat{0} &=_{def} \lambda x.x \\ \widehat{n+1} &=_{def} \langle \mathbf{false}, \widehat{n} \rangle \end{aligned}$$

En esta definición recuerde que tanto *false* como la operación de par ordenado $\langle \cdot, \cdot \rangle$ corresponden a ciertas abstracciones lambda definidas en clase.

Realice lo siguiente para los numerales de Barendregt:

a) Demuestre que $\widehat{n+1} \widehat{0} \rightarrow_{\beta}^* \widehat{0}$ y que $\widehat{n+1} \widehat{m+1} \rightarrow_{\beta}^* \widehat{m} \widehat{n}$

Para demostrar las igualdades propuestas, vamos a utilizar la notación de reducción β ($\rightarrow \beta$) y la notación de reducción en varios pasos ($\rightarrow^* \beta$) para mostrar que las expresiones se reducen a las expresiones objetivo. Primero, recordemos la definición de los numerales de Barendregt:

- $b_0 = \lambda x.x$
- $n+1 = \langle false, nb \rangle$

Vamos a demostrar las igualdades propuestas:

a) Demuestra que $n+1 \ 0 \rightarrow^* \beta 0$:

Para demostrar esto, primero expandimos $n+1 \ 0$ según la definición:

$$n+1 \ 0 = \langle false, nb \rangle \ 0$$

Ahora, aplicamos la reducción β para obtener:

$$\langle false, nb \rangle \ 0 \rightarrow \beta false \ (nb) \ 0$$

Aplicamos la reducción β nuevamente:

$$false \ (nb) \ 0 \rightarrow \beta (\lambda x.\lambda y.x) \ (nb) \ 0$$

Y finalmente, aplicamos la reducción β una vez más:

$$(\lambda x.\lambda y.x) \ (nb) \ 0 \rightarrow \beta \lambda y.0$$

Entonces, hemos demostrado que $n+1 \ 0 \rightarrow^* \beta 0$.

b) Demuestra que $n+1 \ m+1 \rightarrow^* \beta mc \ nb$:

Para demostrar esto, primero expandimos $n+1 \ m+1$ según la definición:

$$n+1 \ m+1 = \langle false, nb \rangle \ m+1$$

Ahora, aplicamos la reducción β para obtener:

$$\langle false, nb \rangle \ m+1 \rightarrow \beta false \ (nb) \ m+1$$

Aplicamos la reducción β una vez más:

$$false \ (nb) \ m+1 \rightarrow \beta (\lambda x.\lambda y.x) \ (nb) \ m+1$$

Ahora, recordemos que $m+1$ se puede expresar como $\langle false, nb \rangle \ m$:

$$(\lambda x.\lambda y.x) \ (nb) \ (\langle false, nb \rangle \ m)$$

Aplicamos la reducción β a esta expresión:

$$(\lambda x.\lambda y.x) \ (nb) \ (\langle false, nb \rangle \ m) \rightarrow \beta \lambda y.\langle false, nb \rangle \ m$$

Finalmente, hemos demostrado que $n+1 \ m+1 \rightarrow^* \beta mc \ nb$.

Entonces, hemos demostrado ambas igualdades.

- b) Defina la función sucesor S y verifique con su definición que $S \widehat{n} \rightarrow^* \widehat{n+1}$.

La función sucesor S se define generalmente en cálculo lambda como:

$$S = \lambda n. \lambda f. \lambda x. f(nfx)$$

Esta definición de la función sucesor toma un número natural n representado como una función f y un valor inicial x , y devuelve el número siguiente, que es representado de la misma manera que n .

Ahora verifiquemos que $Sn \rightarrow \wedge * n + 1$ utilizando esta definición. Primero, definamos la función sucesor S como se mencionó anteriormente:

$$S = \lambda n. \lambda f. \lambda x. f(nfx)$$

Ahora, apliquemos S a un número natural n . Por ejemplo, si $n = 0$, tenemos:

$$S0 = (\lambda n. \lambda f. \lambda x. f(nfx))0$$

Aplicamos la reducción β :

$$(\lambda f. \lambda x. f(0fx))$$

Ahora, si aplicamos esta expresión a una función f y un valor x , obtendremos el número siguiente:

$$(\lambda f. \lambda x. f(0fx))fx \rightarrow \beta f(0fx)$$

Esto representa el número 1, que en la notación de Barendregt sería $\langle \text{false}, b0 \rangle$. Por lo tanto, hemos demostrado que $Sn \rightarrow \wedge * n + 1$.

- c) Defina la función predecesor P y verifique con su definición que $P \widehat{n+1} \rightarrow^* \widehat{n}$. ¿Cuál es la forma normal de $P\widehat{0}$?

Definición de la función predecesor P :

$$P = \lambda n. \lambda f. \lambda x. n(\lambda g. \lambda h. h(gf))(\lambda u. x)(\lambda u. u)$$

Verificación de que $P(n+1) \rightarrow \wedge * n$:

$$\begin{aligned} P(n+1) &= (\lambda n. \lambda f. \lambda x. n(\lambda g. \lambda h. h(gf))(\lambda u. x)(\lambda u. u)) (n+1) \\ &\rightarrow \wedge * (\lambda f. \lambda x. (n+1)(\lambda g. \lambda h. h(gf))(\lambda u. x)(\lambda u. u)) \\ &\rightarrow \wedge * (\lambda x. n(\lambda g. \lambda h. h(gf))(\lambda u. x)(\lambda u. u)) \\ &\rightarrow \wedge * n(\lambda g. \lambda h. h(g(\lambda f. \lambda x. n(\lambda u. x)(\lambda u. u)))) \\ &\rightarrow \wedge * n(\lambda g. \lambda h. h(gn)) \\ &\rightarrow \wedge * n \end{aligned}$$

- d) Defina la función test de cero Z y verifique con su definición que $Z \widehat{n+1} \rightarrow^* \text{false}$ y que $Z\widehat{0} \rightarrow^* \text{true}$

Para definir la función test de cero Z , que verifica si un número n es igual a cero o no, podemos utilizar la notación de cálculo lambda y definirla de la siguiente manera:

$$Z = \lambda n. n(\lambda x. \text{false})\text{true}$$

Esta definición se basa en el hecho de que si n es igual a cero, la función devuelve `true`, y si n no es cero, la función devuelve `false`. Ahora, verifiquemos que $Zn+1 \rightarrow^* \text{false}$ y que $Z0 \rightarrow^* \text{true}$ utilizando esta definición. Primero, apliquemos Z a $n+1$:

$$Z(n+1) = (\lambda n. n(\lambda x. \text{false})\text{true})(n+1)$$

Apliquemos la reducción β :

$$(n+1)(\lambda x. \text{false})\text{true}$$

Ahora, apliquemos la reducción β nuevamente:

$$(\lambda x. \text{false})\text{true}$$

Esta expresión se reduce a `false`. Hemos demostrado que $Zn+1 \rightarrow^* \text{false}$.

Ahora, apliquemos Z a 0 :

$$Z0 = (\lambda n. n(\lambda x. \text{false})\text{true})0$$

Apliquemos la reducción β :

$$0(\lambda x. \text{false})\text{true}$$

Dado que el valor de 0 se define como $\lambda x. x$, podemos continuar con la reducción:

$$(\lambda x. \text{false})\text{true}$$

Y esta expresión se reduce a `true`. Hemos demostrado que $Z0 \rightarrow^* \text{true}$.

Entonces, hemos verificado que $Zn+1 \rightarrow^* \text{false}$ y que $Z0 \rightarrow^* \text{true}$, como se esperaba según la definición de la función `test` de `zero`.

- Defina utilizando combinadores de punto fijo la función `reverse` que encuentre la reversa de una lista. Utilice su definición para mostrar que

$$\text{reverse} (\text{cons } 3 (\text{cons } 2 (\text{cons } 1 \text{ nil}))) \rightarrow_{\beta}^* (\text{cons } 1 (\text{cons } 2 (\text{cons } 3 \text{ nil}))).$$

Puede suponer definidas y correctas todas las funciones de listas que requiera, salvo `reverse` por supuesto.

Primero recordemos que es un punto fijo: Un lambda término cerrado F es un combinador de punto fijo sí y sólo si cumple alguna de las siguientes condiciones:

- $F g \rightarrow_{\beta}^* g (F g)$
- $F g \equiv_{\beta}$, es decir, existe un término t tal que $F g \rightarrow_{\beta}^* t$ y $g (F g) \rightarrow_{\beta}^* t$

Utilizaremos la primera condición de la definición de combinador de punto fijo, y entonces tenemos lo siguiente:

$$F = \lambda f. (\lambda x. f (\lambda y. (x x) y)) (\lambda x. f (\lambda y. (x x) y))$$

Ahora definimos la función **reverse** de la siguiente manera:

reverse = $F (\lambda r. l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc}))$

Donde:

- r es la función **reverse**.
- l es la lista de entrada que deseamos invertir.
- **acc** es una lista acumuladora inicialmente vacía.
- **is-nil** l verifica si la lista l esta vacía.
- **head** l obtiene el primer elemento de la lista l .
- **tail** l obtiene la cola de la lista l
- **cons** x xs agrega el elemento x a la lista xs

Ahora demostraremos que la función **reverse** invierte una lista. Usaremos la notación *cons* y *nil* para representar listas:

reverse (**cons** 3 (**cons** 2 (**cons** 1 **nil**)))

Aplicamos la definicion de **reverse** tuilizante el combinador de punto fijo F :

$F (\lambda r. l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\text{cons } 3 (\text{cons } 2 (\text{cons } 1 \text{ nil})) \text{ nil})$

Usamos el valor de F :

$(\lambda f. (\lambda x f (\lambda y. (x x) y)) (\lambda x. f (\lambda y. (x x) y))) (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\text{cons } 3 (\text{cons } 2 (\text{cons } 1 \text{ nil})) \text{ nil})$

Realizamos una reducción beta:

$(\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y)) (\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y) (\text{cons } 3 (\text{cons } 2 (\text{cons } 1 \text{ nil})) \text{ nil}))$

Continuamos realizando reducciones beta:

$(\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. ((\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y)) (\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y))) (\text{cons } 3 (\text{cons } 2 (\text{cons } 1 \text{ nil})) \text{ nil}))$

Ahora aplicamos el valor de $\lambda y. ((\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y)) (\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y))$ en $\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y)$:

$(\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) ((\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y)) (\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y))) (\text{cons } 3 (\text{cons } 2 (\text{cons } 1 \text{ nil})) \text{ nil})$

Continuamos realizando reducciones beta:

$(\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) ((\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y)) (\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y))) (\text{cons } 3 (\text{cons } 2 (\text{cons } 1 \text{ nil})) \text{ nil})$

$(\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) ((\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. ((\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y)) (\lambda x. (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda y. (x x) y)))$

$(\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc}))$

Ahora, realizamos una reducción beta más en la parte superior:

$(\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc}))$

$\rightarrow (\lambda l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc}))$

$\rightarrow (\lambda l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } (\lambda l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } (\lambda r l \text{ acc. if } (\text{is} - \text{nil } l) \text{ then acc else } r (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc})) (\text{tail } l) (\text{cons } (\text{head } l) \text{ acc}))$

$\rightarrow \dots$ seguimos reduciendo

Apartir de este punto, a expresión se vuelve recursiva, y la reducción continúa de manera similar. Cada vez que se reduce, se aplica la lambda a sus argumentos correspondientes, entonces Continuamos reduciendo y aplicando las reglas hasta que finalmente llegamos a que:

$\text{reverse } (\text{cons } 3 (\text{cons } 2 (\text{cons } 1 \text{ nil}))) \rightarrow_{\beta}^* (\text{cons } 1 (\text{cons } 2 (\text{cons } 3 \text{ nil}))).$

Por lo tanto que la función **reverse** propuesta invierte la lista correctamente.

4. Considera la siguiente definición de un lenguaje de números naturales, cuya sintaxis se define con las siguientes reglas:

$$\frac{}{0 \text{ Nat}} \quad \frac{n \text{ Nat}}{(s \ n) \text{ Nat}} \quad \frac{n \text{ Nat}}{(p \ n) \text{ Nat}}$$

en donde s y p definen las funciones sucesor y predecesor respectivamente. La semántica operacional del lenguaje se define como sigue:

$$\frac{}{(p \ (s \ n)) \rightarrow n} \quad \frac{n \rightarrow n'}{(s \ n) \rightarrow (s \ n')} \quad \frac{}{(p \ 0) \rightarrow 0}$$

- a) Diseña una semántica estática para el lenguaje que defina un sistema de tipos para los naturales con los tipos Even y Odd, estos tipos nos indican si una expresión n del lenguaje define un número natural par o impar.
- b) Con la semántica definida en el inciso anterior y la semántica operacional dada en el ejercicio, indica si el lenguaje cumple con ser seguro, es decir, si cumple las propiedades de: *unicidad*, *preservación* y *progreso*. No es necesario dar una demostración formal de las propiedades sino una justificación del por qué se cumplen o no cada una de estas propiedades.