Universdad Nacional Autónoma de México

**Lenguajes de Programacion**
**Examen Parcial I**


- Edgar Montiel Ledesma
  317317794
- Carlos Daniel Cortes Jimenez
  420004846


---

1. Chon Hacker quiere 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:

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

   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 $\boldsymbol{e}$ **correct** que indica que la expresión $\boldsymbol{e}$ está correctamente en notación polaca, en seguida tenemos nustras reglas de inferencia:

   1.-Si $\boldsymbol{e}$ es un número, entonces:

   $$\frac{}{n \ \textbf{correct}}$$

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

   $$\frac{e_1 \ \textbf{correct} \qquad e_2 \ \textbf{correct}}{+ \ \textbf{correct}} \qquad \frac{e_1 \ \textbf{correct} \qquad e_2 \ \textbf{correct}}{* \ \textbf{correct}} \qquad \frac{e_1 \ \textbf{correct} \qquad e_2 \ \textbf{correct}}{- \ \textbf{correct}}$$

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

$$\frac{}{\epsilon \textbf{ correct}}$$

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 sematica operacional de paso grande es la sigueinte:

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

$$\frac{}{n \Downarrow n}$$

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($\epsilon$):Una expresión vacía no tiene valor

$$\frac{}{\epsilon \Downarrow error}$$

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 ($\epsilon$), 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 espreciones vacías($\epsilon$):

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

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 lenguajes $\circ \models e \rightarrow^*_{rpn} \circ \models v$ 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 $\circ \models sym\ e_1\ rpn \rightarrow^*_{rpn} \circ\ op(sym, v_1, v_2) = \circ \models v$.

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 evalución. 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:

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

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 $\widehat{n+1}\,\widehat{0} \rightarrow^{\star}_{\beta} \widehat{0}$ y que $\widehat{n+1}\,\widehat{m+1} \rightarrow^{\star}_{\beta} \widehat{m}\,\widehat{n}$

Para demostrar las igualdades propuestas, vamos a utilizar la notación de reducción $\beta$ ($\rightarrow \beta$) 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:

- $b_0 = \lambda x.x$
- $n+1 = <false, nb>$

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:

$$n+1\ 0 = <false, nb>\ 0$$

Ahora, aplicamos la reducción $\beta$ para obtener:

$$<false, nb>\ 0 \rightarrow \beta false\ (nb)\ 0$$

Aplicamos la reducción $\beta$ nuevamente:

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

Y finalmente, aplicamos la reducción $\beta$ 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:

$$n+1\ m+1 = <false, nb>\ m+1$$

Ahora, aplicamos la reducción $\beta$ para obtener:

$$<false, nb>\ m+1 \rightarrow \beta false\ (nb)\ m+1$$

Aplicamos la reducción $\beta$ 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 $<false, nb>\ m$:

$$(\lambda x.\lambda y.x)\ (nb)\ (<false, nb>\ m)$$

Aplicamos la reducción $\beta$ a esta expresión:

$$(\lambda x.\lambda y.x)\ (nb)\ (<false, nb>\ m) \rightarrow \beta \lambda y.<false, nb>\ m$$

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 $S\,\widehat{n} \to^\star \widehat{n+1}$.

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 $Sn \to \wedge * n + 1$ 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 $\beta$:

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

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

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

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

*c*) Defina la función predecesor $P$ y verifique con su definición que $P\,\widehat{n+1} \to^\star \widehat{n}$. ¿Cúal 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 $P\ (n+1) \to \wedge * n$:

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

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

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 $Zn+1 \rightarrow^*$ false y que $Z0 \rightarrow^*$ 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 $\beta$:

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

Ahora, apliquemos la reducción $\beta$ nuevamente:

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

Esta expresión se reduce a false. Hemos demostrado que $Zn+1 \rightarrow^*$ false.
Ahora, apliquemos $Z$ a 0:

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

Apliquemos la reducción $\beta$:

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

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 $Z0 \rightarrow^*$ true.

Entonces, hemos verificado que $Zn+1 \rightarrow^*$ false y que $Z0 \rightarrow^*$ true, como se esperaba según la definición de la función test de cero $Z$.

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

$$\texttt{reverse}\ (\texttt{cons}\ 3\ (\texttt{cons}\ 2\ (\texttt{cons}\ 1\ \texttt{nil}))) \rightarrow^\star_\beta\ (\texttt{cons}\ 1\ (\texttt{cons}\ 2\ (\texttt{cons}\ 3\ \texttt{nil}))).$$

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:

1. $F\ g \rightarrow^\star_\beta\ g\ (F\ g)$
2. $F\ g \equiv_\beta$, es decir, existe un término $t$ tal que $F\ g \rightarrow^\star_\beta\ t$ y $g\ (F\ g) \rightarrow^\star_\beta\ t$

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

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

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

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

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:

$$\texttt{reverse}\;(\texttt{cons}\;3\;(\texttt{cons}\;2\;(\texttt{cons}\;1\;\texttt{nil})))$$

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

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

Usamos el valor de $F$:

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

Realizamos una reducción beta:

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

Continuamos realizando reducciones beta:

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

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

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

Continuamos realizando reducciones beta:

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

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

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

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

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

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

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

$\rightarrow$ ...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:

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

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:

$$\frac{}{0\ \texttt{Nat}} \qquad \frac{n\ \texttt{Nat}}{(s\ n)\ \texttt{Nat}} \qquad \frac{n\ \texttt{Nat}}{(p\ n)\ \texttt{Nat}}$$

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

$$\frac{}{(p\ (s\ n)) \to n} \qquad \frac{n \to n'}{(s\ n) \to (s\ n')} \qquad \frac{}{(p\ 0) \to 0}$$

*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. Definimos dos tipos: Even para números naturales pares y Odd para números naturales impares. Las reglas de tipado son las siguientes:

1) Para el número natural cero (0), asignamos el tipo Even.

$$\frac{}{0{:}\mathsf{Even}}$$

2) Para una expresión $(s\ n)$, donde $n$ es de tipo Even, asignamos el tipo Odd.

$$\frac{n{:}\mathsf{Even}}{(s\ n){:}\mathsf{Odd}}$$

3) Para una expresión $(s\ n)$, donde $n$ es de tipo Odd, asignamos el tipo Even.

$$\frac{n{:}\mathsf{Odd}}{(s\ n){:}\mathsf{Even}}$$

4) Para una expresión $(p\ n)$, donde $n$ es de tipo Even, asignamos el tipo Odd.

$$\frac{n{:}\mathsf{Even}}{(p\ n){:}\mathsf{Odd}}$$

5) Para una expresión $(p\ n)$, donde $n$ es de tipo Odd, asignamos el tipo Even.

$$\frac{n{:}\mathsf{Odd}}{(p\ n){:}\mathsf{Even}}$$

Estas reglas de tipado establecen las condiciones bajo las cuales una expresión se considera de tipo Even o Odd. En resumen:

- El número natural cero es de tipo Even.
- Si tomamos el sucesor de un número de tipo Even, el resultado es de tipo Odd.
- Si tomamos el sucesor de un número de tipo Odd, el resultado es de tipo Even.
- Si tomamos el predecesor de un número de tipo Even, el resultado es de tipo Odd.
- Si tomamos el predecesor de un número de tipo Odd, el resultado es de tipo Even.

*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.

Para evaluar si el lenguaje cumple con las propiedades de seguridad (unicidad, preservación y progreso), evaluaremos cada una de ellas en base a las definiciones de la semántica estática y operacional:

1) **Unicidad (Unicity):** En un lenguaje seguro, para cada expresión bien tipada, existe una única forma de reducirla o evaluarla.
   - En este lenguaje, las reglas de tipado nos permiten asignar tipos Even o Odd a las expresiones, lo que significa que cada expresión bien tipada tiene un tipo único asignado de acuerdo a las reglas estáticas.
   - Las reglas de semántica operacional también definen de manera única cómo las expresiones se reducen o evalúan. Por ejemplo, las reglas para el sucesor $(s\ n)$ establecen claramente cómo se reduce: si $n$ es de tipo Even, se convierte en Odd y viceversa. Las reglas para el predecesor $(p\ n)$ también son únicas y deterministas.

   Por lo tanto el lenguaje cumple con la propiedad de unicidad.
2) **Preservación (Preservation):** En un lenguaje seguro, si una expresión tiene un tipo en un paso de reducción, entonces la expresión resultante también tiene ese tipo.
   - Las reglas de tipado están diseñadas para garantizar que la asignación de tipos se preserve durante la reducción. Por ejemplo, si una expresión tiene el tipo Even y se reduce según la regla de sucesor, el resultado tendrá el tipo Odd, lo que demuestra que el tipo se preserva durante la reducción.
   - Las reglas de semántica operacional también se diseñan para preservar los tipos asignados. Por ejemplo, la regla para el predecesor $(p\ (s\ n))$ establece que se reduce a $n$ y, por lo tanto, el tipo se preserva.

   Por lo tanto, el lenguaje cumple con la propiedad de preservación
3) **Progreso (Progress):** En un lenguaje seguro, una expresión bien tipada no se atasca y siempre puede avanzar en la reducción (o finalizar en un valor).
   - En este lenguaje, las reglas de semántica operacional garantizan el progreso. Por ejemplo, la regla para el predecesor $(p\ 0)$ establece que se reduce a 0, que es un valor. La regla para el sucesor $(s\ n)$ garantiza que si $n$ es bien tipado, entonces $(s\ n)$ también puede reducirse a una nueva expresión.
   - Dado que las reglas operacionales aseguran que una expresión bien tipada siempre puede avanzar en la reducción (o llegar a un valor), el lenguaje cumple con la propiedad de progreso.

Por lo que el lenguaje cumple con las propiedades de unicidad, preservación y progreso, lo que lo hace seguro.