Taller Reconocimiento de Patrones

Estudiante: Juan Sebastian Getial Getial

Código: 202124644

Asignatura: Fundamentos de Programación Funcional y Concurrente

Docente: Juan Francisco Diaz Frias

Universidad del Valle

Santiago de Cali

2022

Informe de Uso del Reconocimiento de Patrones

Funciones	**Reconocimiento de Patrones**
mostrar	Si
derivar	Si
evaluar	Si
limpiar	Si
raizNewton	No
buenaAprox	No

En cuanto a la función raizNewton y buenaAprox, cabe recalcar que aunque no utilizan reconocimiento de patrones en su cuerpo, si usan funciones que lo utilizan(derivar y evaluar), y por ello no fue necesario implementarlo directamente en su cuerpo.

Por último, de acuerdo a las funciones hechas, se puede decir que el reconocimiento de patrones es una técnica de programación muy expresiva a la hora de programar, ya que permitió expresar y operar funciones matemáticas de una manera muy versátil y fácil de entender, por lo que es una técnica de programación muy poderosa.

Informe de Corrección

Argumentación Sobre la Corrección

Definiciones(Def)

Para argumentar la corrección de las siguientes funciones tendremos en cuenta las siguientes definiciones:

Expr(Expresiones).

$\forall\ e \in Expr$:

$$
\begin{array}{ll}
 & e\ ==\ Numero(d) \\
\vee & e\ ==\ Atomo(x) \\
\vee & e\ ==\ Suma(e1,\ e2) \\
\vee & e\ ==\ Resta(e1,\ e2) \\
\vee & e\ ==\ Prod(e1,\ e2) \\
\vee & e\ ==\ Div(e1,\ e2) \\
\vee & e\ ==\ Expo(e1,\ e2) \\
\vee & e\ ==\ Logaritmo(e1) \\
\wedge & e1,\ e1 \in Expr\ \wedge\ d \in Double\ \wedge\ x \in Char
\end{array}
$$

Derivada

c	→’	0 (c es constante)	(Def1)
x	→’	1	(Def2)
f + g	→’	f’ + g’	(Def3)
f - g	→’	f’ - g’	(Def4)
f · g	→’	f’ · g + f · g’	(Def5)
$\frac{f}{g}$	→’	$\frac{f' \cdot g - f \cdot g'}{g^2}$	(Def6)
$ln(f)$	→’	$\frac{f'}{f}$	(Def7)
f^g	→’	$f^g * (\frac{f' * g}{f}\ +\ g' \cdot ln(f))$	(Def8)

Función mostrar

Definiciones(Def).

Para argumentar la corrección de esta función tendremos en cuenta la siguientes definiciones:

DefSimb.

$\forall\ e \in Expr$:

$[e\ ==\ Numero(d)] \Leftrightarrow [DefSimb(e)\ ==\ d]$

$\wedge$

$[e\ ==\ Atomo(x)] \Leftrightarrow [DefSimb(e)\ ==\ x]$

$\wedge$

$[e\ ==\ Logaritmo(e1)] \Leftrightarrow [DefSimb(e)\ ==\ (lg(DefSimb(e1)))]$

$\wedge$

$[e\ ==\ Suma(e1,\ e2)] \Leftrightarrow [DefSimb(e)\ ==\ (DefSimb(e1)\ +\ DefSimb(e2))]$

$\wedge$

$[e\ ==\ Resta(e1,\ e2)] \Leftrightarrow [DefSimb(e)\ ==\ (DefSimb(e1)\ -\ DefSimb(e2))]$

$\wedge$

$[e\ ==\ Prod(e1,\ e2)] \Leftrightarrow [DefSimb(e)\ ==\ (DefSimb(e1)\ *\ DefSimb(e2))]$

$\wedge$

$[e\ ==\ Div(e1,\ e2)] \Leftrightarrow [DefSimb(e)\ ==\ (DefSimb(e1)\ /\ DefSimb(e2))]$

$\wedge$

$[e\ ==\ Expo(e1,\ e2)] \Leftrightarrow [DefSimb(e)\ ==\ (DefSimb(e1)\ \hat{}\ DefSimb(e2))]$

$\wedge$

$e1, e2 \in Expr$

Demostración.

A continuación se demostrará que:

$\forall\ e \in Expr$: $mostrar(e)\ ==\ DefSimb(e)$.

En primer lugar, procederemos a demostrar los casos base de la función.

Caso Base 1.

$mostrar(Numero(d))\ \ ==\ \ DefSimb(Numero(d))$

$mostrar(Numero(d))\ \ ==\ \ d$

Dem:

$\rightarrow\quad mostrar(Numero(d))$

$\rightarrow\quad d$ ♦

Caso Base 2.

$mostrar(Atomo(x))\ \ ==\ \ DefSimb(Atomo(x))$

$mostrar(Atomo(x))\ \ ==\ \ x)$

Dem:

$\rightarrow\quad mostrar(Atomo(x))$

$\rightarrow\quad x$ ♦

De acuerdo a estos casos se procederá a demostrar los casos de inducción.

Caso Inducción 1.

En primer lugar, cabe recalcar que las expresiones Suma, Resta, Prod, Div y Expo poseen una estructura prácticamente igual que difiere solamente en un símbolo: +, -, *, / y ^, respectivamente. Por lo que para este caso se generalizara dichos símbolos con un # y al conjunto de expresiones antes mencionado con **OpComp**.

$Hipótesis: mostrar(e1)\ \ ==\ DefSimb(e1)\ \ \wedge\ mostrar(e2)\ ==\ DefSimb(e2)$

$Hipótesis\ \ \Rightarrow mostrar(OpComp(e1,\ e2))\ ==\ DefSimb(OpComp(e1,\ e2))$

$Hipótesis\ \ \Rightarrow mostrar(OpComp(e1,\ e2))\ ==\ (DefSimb(e1)\ \#\ DefSimb(e2))$

Dem:

$\rightarrow\quad mostrar(OpComp(e1,\ e2))$

$\rightarrow\quad (mostrar(e1)\ \#\ mostrar(e2))$

Hip$\rightarrow$ $(DefSimb(e1)\ \#\ DefSimb(e2))$ ♦

Caso Inducción 2.

El último caso a considerar es el de la expresión Logaritmo.

$Hipótesis: mostrar(e1)\ \ ==\ DefSimb(e1)$

$Hipótesis \Rightarrow mostrar(Logaritmo(e1))\ ==\ DefSimb(Logaritmo(e1))$

$Hipótesis \Rightarrow mostrar(Logaritmo(e1)) == (lg(DefSimb(e1)))$

Dem:

→ $mostrar(Logaritmo(e1))$

→ $(lg(mostrar(e1)))$

Hip→ $(lg(DefSimb(e1)))$ ♦

Finalmente, al haber demostrado todos los posibles casos de la función y teniendo en cuenta que es un patrón se tiene que:

$\forall\, e \in Expr{:}\ mostrar(e) == DefSimb(e)$ ♦

Función derivar

A continuación se demostrará que:

$\forall\, e \in Expr\ \forall\, a \in Atomo{:}\ derivar(e, a) == e'(a)$, donde $e'(a)$ es la definición simbólica de la derivada según el caso.

Note: <u>Tendremos en cuenta que anteriormente en la función mostrar demostramos implícitamente que cada expresión representa su definición simbólica equivalente.</u>

Para ello, en primer lugar, procederemos a demostrar sus casos base:

Caso Base 1:

derivar(Numero(d), Atomo(a)) == 0’ == 0

Dem:

derivar(Numero(d), Atomo(a))

→ Numero(0)

→ 0 ♦

Caso Base 2:

derivar(Atomo(x), Atomo(x)) == x’ = 1

Dem:

derivar(Atomo(x), Atomo(x))

→ if (Atomo(x) == Atomo(x)) Numero(1) else Numero(0)

→ Numero(1)

→ 1 ♦

Caso Base 3:

derivar(Atomo(x), Atomo(z)) == x’ = 0

Dem:

derivar(Atomo(x), Atomo(z))

→ if (Atomo(x) == Atomo(z)) Numero(1) else Numero(0)

→ Numero(0)

→ 0 ♦

Caso Inducción 1(Suma).

$Hipotesis: derivar(e1,\ Atomo(a))\ ==\ e1'(a)\ \wedge\ derivar(e2,\ Atomo(a))\ ==\ e2'(a)$

$Hipotesis \Rightarrow derivar(Suma(e1,\ e2),\ Atomo(a))\ ==\ (e1\ +\ e2)'$

Dem:

$derivar(Suma(e1,\ e2),\ Atomo(a))$

→ $Suma(derivar(e1,\ Atomo(a)),\ derivar(e2,\ Atomo(a)))$

→ $derivar(e1,\ Atomo(a))\ +\ derivar(e2,\ Atomo(a))$

Hip→ $e1'(a)\ +\ e2'(a)$

Def3→$(e1\ +\ e2)'$ ♦

Caso Inducción 2(Resta).

$Hipotesis: derivar(e1,\ Atomo(a))\ ==\ e1'(a)\ \wedge\ derivar(e2,\ Atomo(a))\ ==\ e2'(a)$

$Hipotesis \Rightarrow derivar(Resta(e1,\ e2),\ Atomo(a))\ ==\ (e1\ -\ e2)'$

Dem:

$derivar(Resta(e1,\ e2),\ Atomo(a))$

→ $Resta(derivar(e1,\ Atomo(a)),\ derivar(e2,\ Atomo(a)))$

→ $derivar(e1,\ Atomo(a))\ -\ derivar(e2,\ Atomo(a))$

Hip→ $e1'(a)\ -\ e2'(a)$

Def4→$(e1\ -\ e2)'$ ♦

Caso Inducción 3(Prod).

$Hipotesis: derivar(e1,\ Atomo(a))\ ==\ e1'(a)\ \wedge\ derivar(e2,\ Atomo(a))\ ==\ e2'(a)$

$Hipotesis \Rightarrow derivar(Prod(e1,\ e2),\ Atomo(a))\ ==\ (e1\ *\ e2)'$

Dem:

$derivar(Prod(e1,\ e2),\ Atomo(a))$

$\rightarrow Suma(Prod(derivar(e1,\ Atomo(a)),\ e2),\ Prod(e1,\ derivar(e2,\ Atomo(a))))$

$\rightarrow \quad Prod(derivar(e1,\ Atomo(a)),\ e2) + Prod(e1,\ derivar(e2,\ Atomo(a)))$

$\rightarrow \quad derivar(e1,\ Atomo(a)) * \ e2 + e1 * derivar(e2,\ Atomo(a))$

Hip$\rightarrow$ $e1'(a) * \ e2 + e1 * e2'(a)$

Def5$\rightarrow$$(e1 * e2)'$ ♦

Caso Inducción 4(Div).

$Hipotesis: derivar(e1,\ Atomo(a)) == e1'(a) \wedge derivar(e2,\ Atomo(a)) == e2'(a)$

$Hipotesis \Rightarrow derivar(Div(e1,\ e2),\ Atomo(a)) == (\frac{e1}{e2})'$

Dem:

$derivar(Div(e1,\ e2),\ Atomo(a))$

$\rightarrow \quad Div(Resta($

$Prod(derivar(e1,\ Atomo(a)),\ e2),\ Prod(e1,\ derivar(e2,\ Atomo(a)))),\ Expo(e2,\ Numero(2)))$

$\rightarrow \quad \frac{Resta(Prod(derivar(e1, Atomo(a)), e2), Prod(e1, derivar(e2, Atomo(a))))}{Expo(e2, Numero(2))}$

$\rightarrow \quad \frac{Prod(derivar(e1, Atomo(a)), e2) - Prod(e1, derivar(e2, Atomo(a)))}{e2\,^\wedge\, Numero(2)}$

$\rightarrow \quad \frac{derivar(e1, Atomo(a))* e2 - e1*derivar(e2, Atomo(a))}{(e2)^2}$

Hip$\rightarrow$ $\frac{e1'(a)* e2 - e1*e2'(a)}{(e2)^2}$

Def6$\rightarrow$$(\frac{e1}{e2})'$ ♦

Caso Inducción 5(Expo).

$Hipotesis: derivar(e1,\ Atomo(a)) == e1'(a) \wedge derivar(e2,\ Atomo(a)) == e2'(a)$

$Hipotesis \Rightarrow derivar(Expo(e1,\ e2),\ Atomo(a)) == (e1^{e2})'$

Dem:

$\rightarrow \quad derivar(Expo(e1,\ e2),\ Atomo(a))$

$\rightarrow \quad Prod(Expo(e1,\ e2),$

$Suma($

$Div(Prod(derivar(e1, Atomo(a)),\ e2),\ e1),\ Prod(derivar(e2, Atomo(a)),\ Logaritmo(e1))))$

$\rightarrow \quad Expo(e1,\ e2)* [Div(Prod(derivar(e1, Atomo(a)),\ e2),\ e1)$

$+ \ Prod(derivar(e2, Atomo(a)), \ Logaritmo(e1))]$

$\rightarrow \quad e1^{e2} * [\frac{Prod(derivar(e1,Atomo(a)), e2)}{e1} + derivar(e2, Atomo(a)) * lg(e1)]$

$\rightarrow \quad e1^{e2} * [\frac{derivar(e1,Atomo(a))*e2}{e1} + derivar(e2, Atomo(a)) * lg(e1)]$

Hip$\rightarrow$ $e1^{e2} * [\frac{e1'(a)*e2}{e1} + e2'(a) * lg(e1)]$

Def8$\rightarrow$$(e1^{e2})'$ ♦

Caso Inducción 6(Logaritmo).

$Hipotesis$: $derivar(e1, \ Atomo(a)) \ == \ e1'(a)$

$Hipotesis \Rightarrow derivar(Logaritmo(e1), \ Atomo(a)) \ == \ (lg(e1))'$

Dem:

$derivar(Logaritmo(e1)), \ Atomo(a))$

$\rightarrow \quad Div(derivar(e1, \ Atomo(a)), \ e1)$

$\rightarrow \quad \frac{derivar(e1, Atomo(a))}{e1}$

Hip$\rightarrow$ $\frac{e1'(a)}{e1}$

Def7$\rightarrow$$(lg(e1))'$ ♦

Finalmente, al haber demostrado todos los posibles casos de la función y teniendo en cuenta que es un patrón se tiene que:

$\forall \ e \in Expr \ \forall \ a \in Atomo$: $derivar(e, \ a) \ == \ e'(a)$, donde $e'(a)$ es la definición simbólica de la derivada según el caso.

Función evaluar

A continuación se demostrará que:

$\forall \ e \in Expr \ \forall \ a \in Atomo \ \forall \ v \in Double$: $evaluar(e, \ a, \ v) \ == \ e(v)$, donde $e(v)$ es el valor resultante de evaluar v en $e(a)$.

En primer lugar demostremos los casos base:

Caso base 1:

$evaluar(Numero(d), \ Atomo(a), \ v)$ == d

Dem:

$evaluar(Numero(d), \ Atomo(a), \ v)$

$\rightarrow \quad d \quad \blacklozenge$

Caso base 2:

$evaluar(Atomo(x),\ Atomo(a),\ v)\ ==\ v$

Dem:

$evaluar(Atomo(x),\ Atomo(a),\ v)$

$\rightarrow \quad v \quad \blacklozenge$

Caso Inducción 1.

En primer lugar, como se mencionó en una demostración anterior Suma, Resta, Prod, Div y Expo poseen una estructura prácticamente igual que difiere solamente en un símbolo. Por lo que para este caso también se generalizara dichos símbolos con un # y al conjunto de expresiones antes mencionado con **OpComp**.

$Hipótesis: evaluar(e1,\ Atomo(a),\ v)\ ==\ e1(v) \wedge evaluar(e2,\ Atomo(a),\ v)\ ==\ e2(v)$

$Hipótesis \Rightarrow evaluar(OpComp(e1,\ e2),\ Atomo(a),\ v)\ ==\ OpComp(v)$

$Hipótesis \Rightarrow evaluar(OpComp(e1,\ e2),\ Atomo(a),\ v)\ ==\ e1(v)\ \#\ e2(v)$

Dem:

$evaluar(OpComp(e1,\ e2),\ Atomo(a),\ v)$

$\rightarrow \quad evaluar(e1,\ Atomo(a),\ v)\ \#\ evaluar(e2,\ Atomo(a),\ v)$

Hip$\rightarrow$ $e1(v)\ \#\ e2(v) \blacklozenge$

Caso Inducción 2.

El último caso a considerar es el de la expresión Logaritmo.

$Hipótesis: evaluar(e1,\ Atomo(a),\ v)\ ==\ e1(v)$

$Hipótesis \Rightarrow evaluar(Logaritmo(e1),\ Atomo(a),\ v)\ ==\ lg(e1(v))$

Dem:

$evaluar(Logaritmo(e1),\ Atomo(a),\ v)$

$\rightarrow \quad math.log(evaluar(e1,\ Atomo(a),\ v))$

$\rightarrow \quad lg(evaluar(e1,\ Atomo(a),\ v))$

Hip$\rightarrow$ $lg(e1(v)) \quad \blacklozenge$

Finalmente, al haber demostrado todos los posibles casos de la función y teniendo en cuenta que es un patrón se tiene que:

$\forall\ e \in Expr\ \forall\ a \in Atomo\ \forall\ v \in Double$: $evaluar(e,\ a,\ v)\ \ ==\ \ e(v)$, donde $e(v)$ es el valor resultante de evaluar v en $e(a)$.

Función limpiar

Función limpiarAux

imagen

A continuación se demostrará que:

$\forall\ e \in Expr$: $limpiarAux(e)\ \ ==\ \ E$, donde E es una expresión equivalente a e con m símbolos, tal que $m\ \le\ n$, donde n es el número de símbolos de e.

Caso Base 1:

$limpiarAux(Numero(d))\ \ ==\ Numero(d)$

Dem:

$limpiarAux(Numero(d))$

→ $Numero(d)$ ♦

Caso Base 2:

$limpiarAux(Atomo(x))\ \ ==\ Atomo(x)$

Dem:

$limpiarAux(Atomo(x))$

→ $Atomo(x)$ ♦

Caso Inducción 1(Suma).

$Hipótesis$: $limpiarAux(e1)\ ==\ E1\ \wedge\ limpiarAux(e2)\ ==\ E2$, donde $e1$ y $e2$ tienen x y y número de símbolos respectivamente, y $E1$ y $E2$ tienen z y w número de símbolos respectivamente, tal que $z\ <\ x\ \wedge\ w\ <\ y$.

$Hipótesis \Rightarrow\ limpiarAux(Suma(e1,\ e2))\ ==\ D$, donde D y $Suma(e1,\ e2)$ tienen n y $(x\ +\ y)$ número de símbolos respectivamente, tal que, $n\ <\ (x\ +\ y)$.

Dem:

→ $limpiarAux(Suma(e1,\ e2))$

→ if (e1 == Numero(0)) limpiarAux(e2) (1)

else if (e2 == Numero(0)) limpiarAux(e1) (2) else Suma(limpiarAux(e1), limpiarAux(e2)) (3)

Viendo cada caso por separado se tiene que:

(1)→ $limpiarAux(e2)$

Hip→ $E2$, w símbolos, tal que $w < y$, y por tanto $w < x + y$ ♦

(2)→ $limpiarAux(e1)$

Hip→ $E1$, z símbolos, tal que $z < x$, y por tanto $z < x + y$ ♦

(3)→ $Suma(limpiarAux(e1),\ limpiarAux(e2))$

→ $limpiarAux(e1) + limpiarAux(e2)$

Hip→ D, n símbolos, donde $n = 1 + z + w$, tal que $z < x$ y $w < y$, por tanto $1 + z + w < x + y$ ♦

Caso Inducción 2(Resta).

$Hipótesis$: $limpiarAux(e1) == E1 \wedge limpiarAux(e2) == E2$, donde $e1$ y $e2$ tienen x y y número de símbolos respectivamente, y $E1$ y $E2$ tienen z y w número de símbolos respectivamente, tal que $z < x \wedge w < y$.

$Hipótesis \Rightarrow limpiarAux(Resta(e1,\ e2)) == D$, donde D y $Resta(e1,\ e2)$ tienen n y $(x + y)$ número de símbolos respectivamente, tal que, $n < (x + y)$.

Dem:

→ $limpiarAux(Resta(e1,\ e2))$

→ if (e1 == Numero(0)) Prod(limpiarAux(e2), Numero(-1)) (1)

else if (e2 == Numero(0)) limpiarAux(e1) (2)

else Resta(limpiarAux(e1), limpiarAux(e2)) (3)

Viendo cada caso por separado se tiene que:

(1)→ $Prod(limpiarAux(e2),\ Numero(-\ 1))$

→ $limpiarAux(e2)\ *\ Numero(-\ 1)$

Hip→ D, n símbolos, $n = w + 1$, tal que $w < y$, por lo que $w + 1 \leq x + y$

En este caso se utiliza el ≤ debido a puede que no se reduzca el número de símbolos. Mas sin embargo, no afecta a la demostración. ♦

(2)→ $limpiarAux(e1)$

→ $E1$, z símbolos, tal que $z < x$, por lo que $z < x + y$ ♦

(3)→ $Resta(limpiarAux(e1),\ limpiarAux(e2))$

→ $limpiarAux(e1) - limpiarAux(e2)$

→ D, n símbolos, $n = z + w + 1$,tal que $z < x$ y $w < y$, por lo que $z + w + 1 < x + y$ ♦

Caso Inducción 3(Prod).

$Hipótesis$: $limpiarAux(e1) == E1 \land limpiarAux(e2) == E2$, donde $e1$ y $e2$ tienen x y y número de símbolos respectivamente, y $E1$ y $E2$ tienen z y w número de símbolos respectivamente, tal que $z < x \land w < y$.

$Hipótesis \Rightarrow limpiarAux(Prod(e1, e2)) == D$, donde D y $Prod(e1, e2)$ tienen n y $(x + y)$ número de símbolos respectivamente, tal que, $n < (x + y)$.

Dem:

→ $limpiarAux(Resta(e1, e2))$

→ if (e1 == Numero(0) || e2 == Numero(0)) Numero(0) (1)

else if (e1 == Numero(1)) limpiarAux(e2) (2)

else if (e2 == Numero(1)) limpiarAux(e1) (3)

else Prod(limpiarAux(e1), limpiarAux(e2)) (4)

Viendo cada caso por separado se tiene que:

(1)→ $Numero(0)$

→ D, n símbolos, $n = 0$, por lo que $n < x + y$ ♦

(2)→ $limpiarAux(e2)$

→ $E2$, w símbolos, tal que $w < y$, por lo que $z < x + y$ ♦

(3)→ $limpiarAux(e1)$

→ $E1$, z símbolos, tal que $z < x$, por lo que $z < x + y$ ♦

(4)→ $Prod(limpiarAux(e1), limpiarAux(e2))$

→ $limpiarAux(e1) * limpiarAux(e2)$

→ D, n símbolos, $n = z + w + 1$,tal que $z < x$ y $w < y$, por lo que $z + w + 1 < x + y$ ♦

Caso Inducción 4(Div).

$Hipótesis$: $limpiarAux(e1) == E1 \land limpiarAux(e2) == E2$, donde $e1$ y $e2$ tienen x y y número de símbolos respectivamente, y $E1$ y $E2$ tienen z y w número de símbolos respectivamente, tal que $z < x \land w < y$.

$Hipótesis \Rightarrow\ limpiarAux(Div(e1,\ e2))\ ==\ D$, donde D y $Div(e1,\ e2)$ tienen n y $(x\ +\ y)$ número de símbolos respectivamente, tal que, $n\ <\ (x\ +\ y)$.

Dem:

→ $limpiarAux(Div(e1,\ e2))$

→ if (e1 == Numero(0)) Numero(0) (1)

else if (e2 == Numero(1)) limpiarAux(e1) (2)

else Div(limpiarAux(e1), limpiarAux(e2)) (3)

Viendo cada caso por separado se tiene que:

(1)→ $Numero(0)$

→ D, n símbolos, $n\ =\ 0$, por lo que $n\ <\ x\ +\ y$ ♦

(2)→ $limpiarAux(e1)$

→ $E1$, z símbolos, tal que $z\ <\ x$, por lo que $z\ <\ x\ +\ y$ ♦

(3)→ $Div(limpiarAux(e1),\ limpiarAux(e2))$

→ $limpiarAux(e1)\ /\ limpiarAux(e2)$

→ D, n símbolos, $n\ =\ z\ +\ w\ +\ 1$,tal que $z\ <\ x$ y $w\ <\ y$, por lo que $z\ +\ w\ +\ 1\ <\ x\ +\ y$ ♦

Caso Inducción 5(Expo).

$Hipótesis$: $limpiarAux(e1)\ ==\ E1\ \wedge\ limpiarAux(e2)\ ==\ E2$, donde $e1$ y $e2$ tienen x y y número de símbolos respectivamente, y $E1$ y $E2$ tienen z y w número de símbolos respectivamente, tal que $z\ <\ x\ \wedge\ w\ <\ y$.

$Hipótesis \Rightarrow\ limpiarAux(Expo(e1,\ e2))\ ==\ D$, donde D y $Expo(e1,\ e2)$ tienen n y $(x\ +\ y)$ número de símbolos respectivamente, tal que, $n\ <\ (x\ +\ y)$.

Dem:

→ $limpiarAux(Expo(e1,\ e2))$

→ if (e1 == Numero(0)) Numero(0) (1)

else if (e1 == Numero(1) || e2 == Numero(0)) Numero(1) (2)

else if (e2 == Numero(1)) limpiarAux(e1) (3)

else Expo(limpiarAux(e1), limpiarAux(e2)) (4)

Viendo cada caso por separado se tiene que:

(1)→ $Numero(0)$

→ D, n símbolos, $n\ =\ 0$, por lo que $n\ <\ x\ +\ y$ ♦

(2)→ $Numero(1)$

→ D, n símbolos, $n = 0$, por lo que $n < x + y$ ♦

(3)→ $limpiarAux(e1)$

→ $E1$, z símbolos, tal que $z < x$, por lo que $z < x + y$ ♦

(4)→ $Expo(limpiarAux(e1), limpiarAux(e2))$

→ $limpiarAux(e1)$ ^ $limpiarAux(e2)$

→ D, n símbolos, $n = z + w + 1$,tal que $z < x$ y $w < y$, por lo que $z + w + 1 < x + y$ ♦

Caso Inducción 6(Logaritmo).

$Hipótesis$: $limpiarAux(e1) == E1 \wedge limpiarAux(e2) == E2$, donde $e1$ y $e2$ tienen x y y número de símbolos respectivamente, y $E1$ y $E2$ tienen z y w número de símbolos respectivamente, tal que $z < x \wedge w < y$.

$Hipótesis \Rightarrow limpiarAux(Logaritmo(e1, e2)) == D$, donde D y $Logaritmo(e1, e2)$ tienen n y $(x + y)$ número de símbolos respectivamente, tal que, $n < (x + y)$.

Dem:

→ $limpiarAux(Logaritmo(e1, e2))$

→ $Logaritmo(limpiarAux(e1))$

→ $lg(limpiarAux(e1))$

→ D, n símbolos, $n = z + 1$,tal que $z < x$, por lo que $z + 1 \leq x + y$

En este caso se utiliza el ≤ debido a puede que no se reduzca el número de símbolos. Mas sin embargo, no afecta a la demostración. ♦

Finalmente, al haber demostrado todos los posibles casos de la función y teniendo en cuenta que es un patrón se tiene que:

$\forall\ e \in Expr$: $limpiarAux(e) == E$, donde E es una expresión equivalente a e con m símbolos, tal que $m \leq n$, donde n es el número de símbolos de e.

Demostración.

Teniendo en cuenta lo anterior, se demostrará que:

$\forall\ e \in Expr$: $limpiar(e) == E$, donde E es una expresión equivalente a e con m símbolos, tal que $m \leq n$, donde n es el número de símbolos de e.

Dem:

→ $limpiar(e)$

→ val expresionLimpia = limpiarAux(f)
if(expresionLimpia == f) expresionLimpia (1)
else limpiar(expresionLimpia) (2)

(1)→ expresionLimpia, como se demostró que limpiarAux(e) da como resultado una expresión equivalente con un número menor de símbolos, si el número de símbolos no cambia, significa que la expresión ya no se puede limpiar más, por lo que ese será el resultado más limpio. ♦

(2)→ limpiar(expresionLimpia), de lo contrario se volverá a limpiar. Y esto se repetirá recursivamente hasta que ya no se pueda limpiar más.

Por lo que finalmente se tiene que:

$\forall\ e \in Expr$: $limpiar(e)\ ==\ E$, donde E es una expresión equivalente a e con m símbolos, tal que $m \leq n$, donde n es el número de símbolos de e.

Función raizNewton

Definición.

Raíz(f) - Método de Newton

Este método parte de una aproximación inicial x_{i-1} y obtiene una aproximación mejor, x_i, dada por la fórmula:

$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$, donde f es la función a la cual se le quiere hallar la raíz.

Demostración.

A continuación de demostrar que:

$\forall\ e \in Expr\ \forall\ a \in Atomo\ \forall\ x0 \in Double$: $raizNewton(e,\ a,\ x_i,\ ba)\ \approx\ raíz(e)$, donde

$ba\ \Rightarrow\ buenaAprox(f\!: Expr,\ a\!: Atomo,\ d\!: Double)\ =\ f(d)\ <\ 0.001$

- $Estado(S_i)$: $(e,\ a,\ x_i,\ ba)$
- $Estado\ Inicial(S_0)$: $(e,\ a,\ x_0,\ ba)$
- $Estado\ Final(S_f)$: $(e,\ a,\ x,\ ba)\ \wedge\ e(x)\ <\ 0.001$

- $Respuesta(S_f): x$
- $Trans(S_i)$: $(e,\ a,\ x_{i-1}\ -\ \frac{e(x_{i-1})}{e'(x_{i-1})},\ ba)$
- $Inv(S_i)$: $x_i < x_{i-1}$

1.Inv(**S_1**)

Dem:

$\rightarrow$ Inv$(e,\ a,\ x_1,\ ba)$

$\rightarrow$ $(e,\ a,\ x_0\ -\ \frac{e(x_0)}{e'(x_0)},\ ba)$

$\rightarrow$ $x_1 < x_0$

$\rightarrow$ $x_0\ -\ \frac{e(x_0)}{e'(x_0)} < x_0$

$\rightarrow$ $-\frac{e(x_0)}{e'(x_0)} <\ 0$ ♦

2.$S_i \neq S_f\ \wedge\ Inv(S_i) \rightarrow Inv(Trans(S_i))$

$Hipótesis: e(x_i)\ \geq\ 0.001\ \wedge\ Inv(e,\ a,\ x_i,\ ba)$

$Hipótesis: e(x_i)\ \geq\ 0.001\ \wedge x_i < x_{i-1}$

$Hipótesis \Rightarrow Inv(Trans(S_i))$

$Hipótesis \Rightarrow Inv((e,\ a,\ x_{i-1}\ -\ \frac{e(x_{i-1})}{e'(x_{i-1})},\ ba))$

Dem:

$Inv((e,\ a,\ x_{i-1}\ -\ \frac{e(x_{i-1})}{e'(x_{i-1})},\ ba))$

$\rightarrow$ $x_{i-1}\ -\ \frac{e(x_{i-1})}{e'(x_{i-1})} < x_{i-1}$

$\rightarrow$ $-\ \frac{e(x_{i-1})}{e'(x_{i-1})} <\ 0$ ♦

3.$Inv(S_f) \rightarrow Respuesta(S_f)\ ==\ raíz(e)$

$Hipótesis: Inv(e,\ a,\ x,\ ba)$

$Hipótesis: e(x) < 0.001$

$Hipótesis \Rightarrow Respuesta(S_f) == raíz(e)$

Dem:

$Respuesta(S_f)$

$\rightarrow$ $\boldsymbol{x}$

Hip$\rightarrow$ $x \approx raíz(e)$ ♦

Por otro lado, en cada paso la componente x_i del estado es más pequeña. Por tanto, está cada vez más cerca de 0. En consecuencia, después de n iteraciones llega a ser menor que 0.001. Esto implica que:

$raizNewton(e, a, x_i, ba) == x == raiz(e)$

Por lo que finalmente se tiene que:

$\forall\, e \in Expr\ \forall\, a \in Atomo\ \forall\, x0 \in Double$: $raizNewton(e, a, x_i, ba) \approx raíz(e)$, donde $ba \Rightarrow buenaAprox(f: Expr, a: Atomo, d: Double) = f(d) < 0.001$

Casos de Prueba

Nota: Para no extender demasiado el documento, no se pondrán los casos de prueba. Por favor revisarlos en el código fuente.

En términos generales, en todos los casos de prueba se intentó utilizar una gran parte de las combinaciones posibles de expresiones, sin embargo, aunque no se utilizaron todas, al ser funciones que trabajan con reconocimiento de patrones, se espera, como se demostró en la argumentación de corrección, que funcione para todas las combinaciones posibles.

Por último, en todos los casos de prueba de cada función se obtuvieron resultados acordes al resultado esperado, aunque cabe resaltar que, dependiendo de la estructura de la expresión que recibe la función raizNewton, el resultado obtenido por parte de esta podía ser más o menos aproximado, pero, teniendo en cuenta que solo se esperaban aproximaciones desde un comienzo por parte de esta función, ya que se utilizó la fórmula de Newton, se podría decir que en general, todos los casos de prueba refuerzan la corrección de su respectivo programa.

Conclusiones

Luego de analizar de cada una de las funciones realizadas, se pudo concluir que el reconocimiento de patrones es una herramienta muy expresiva y por consiguiente poderosa a la hora de programar, ya que permite abstraer problemas complejos de una manera sencilla y fácil de entender, lo cual es un indicador de que es una herramienta muy efectiva.

Por último, cabe resaltar que, el problema realizado en este taller no tuvo en cuenta expresiones trigonométricas u otro tipo de expresiones, sin embargo, esto no significa que no se puedan solucionar a través de este método. Ya que simplemente bastaría con añadir la expresión y su respectivo patrón para cada función.