Programación Evolutiva

Tema 7: Programación genética

Carlos Cervigón 2023-2024

Programación genética (John Koza)

Algoritmo evolutivo en el que las estructuras que evolucionan son **árboles** que codifican expresiones o “programas”.

Los individuos o cromosomas se representan en forma de (árbol)

- Los nodos representan funciones
- Las hojas representan símbolos terminales

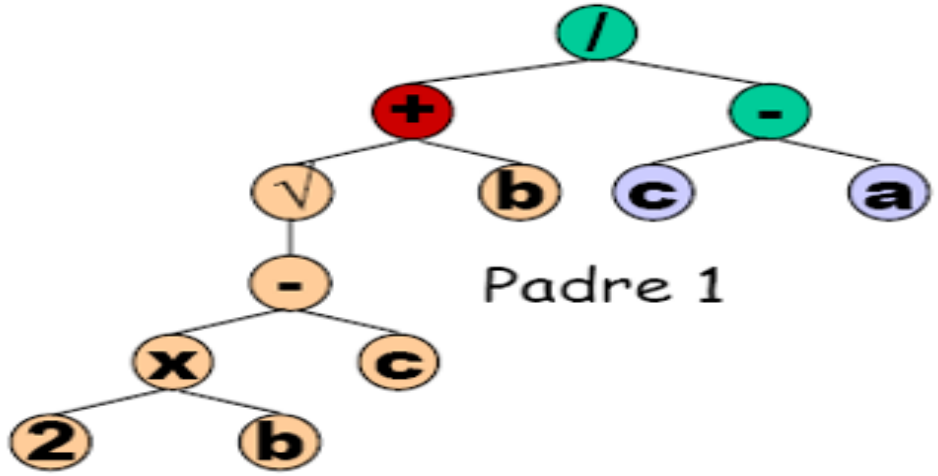
Todos los programas en la población inicial han de ser sintácticamente correctos (ejecutables)

Los operadores genéticos (cruce, mutación, …) también han de producir programas sintácticamente correctos

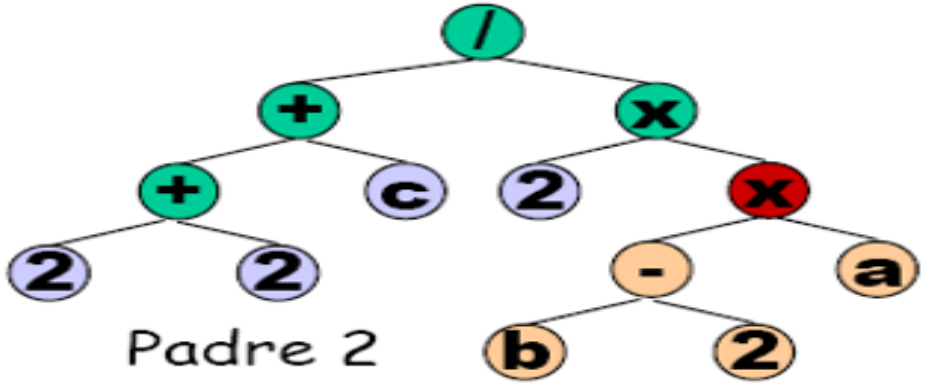
Programación genética: esquema

- Se genera un población inicial de programas aleatorios (árboles) usando el conjunto de funciones y terminales posibles.
  - Sintácticamente correctos y Limitados en profundidad.
- Se aplican los operadores de selección, cruce y mutación durante cierto número de generaciones.
  - Operador de Cruce: Intercambiar dos subárboles (elegidos aleatoriamente) entre los dos árboles padres.
  - Operador de Mutación: por ejemplo, cambiar un nodo terminal o un nodo función

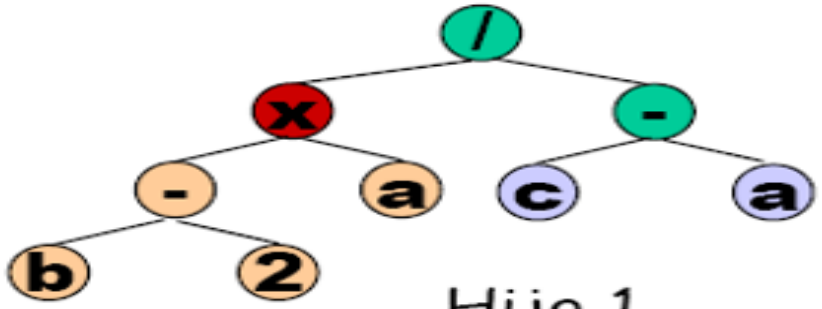
Ejemplo cruce



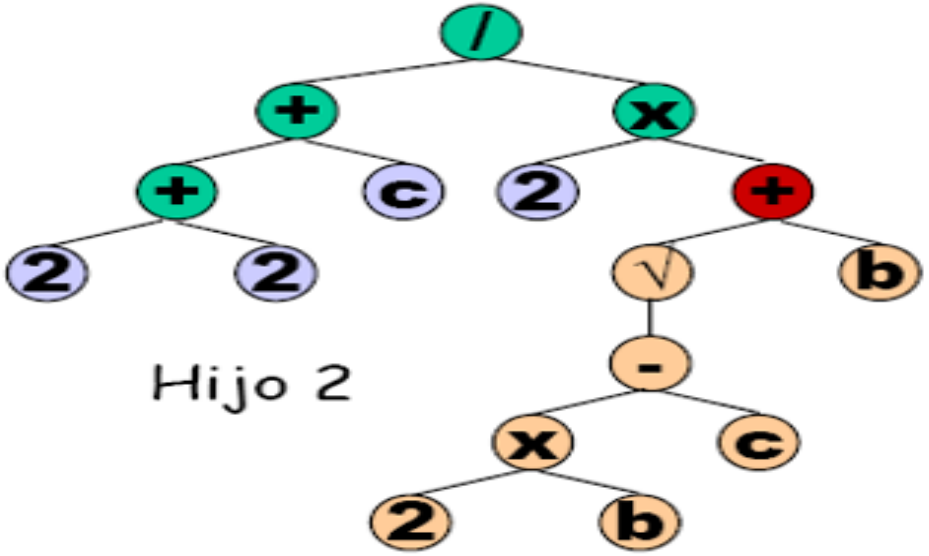
Padre 1



Padre 2



Hijo 1



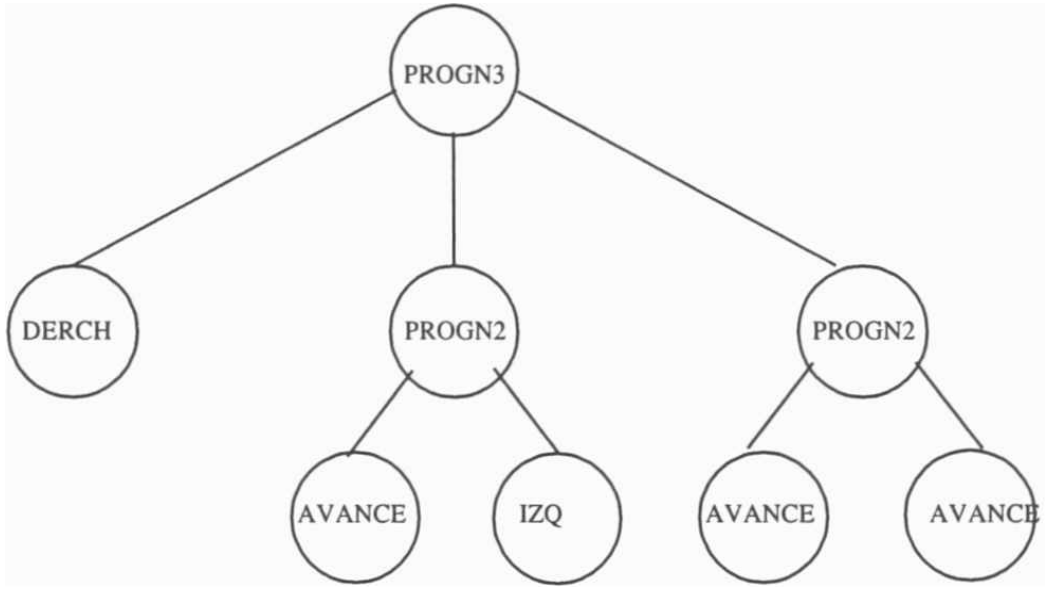
Hijo 2

Algoritmo

- Generar una población inicial de expresiones o programas aleatorios, mediante árboles formados por funciones y símbolos terminales.
- Repetir hasta Fin:
  - Ejecutar cada programa y evaluar su aptitud para resolver el problema.
  - Aplicar selección.
  - Aplicar cruce
  - Aplicar mutación.
- Devolver el mejor programa como solución.

Ejemplo de individuo

Representación en forma de árbol

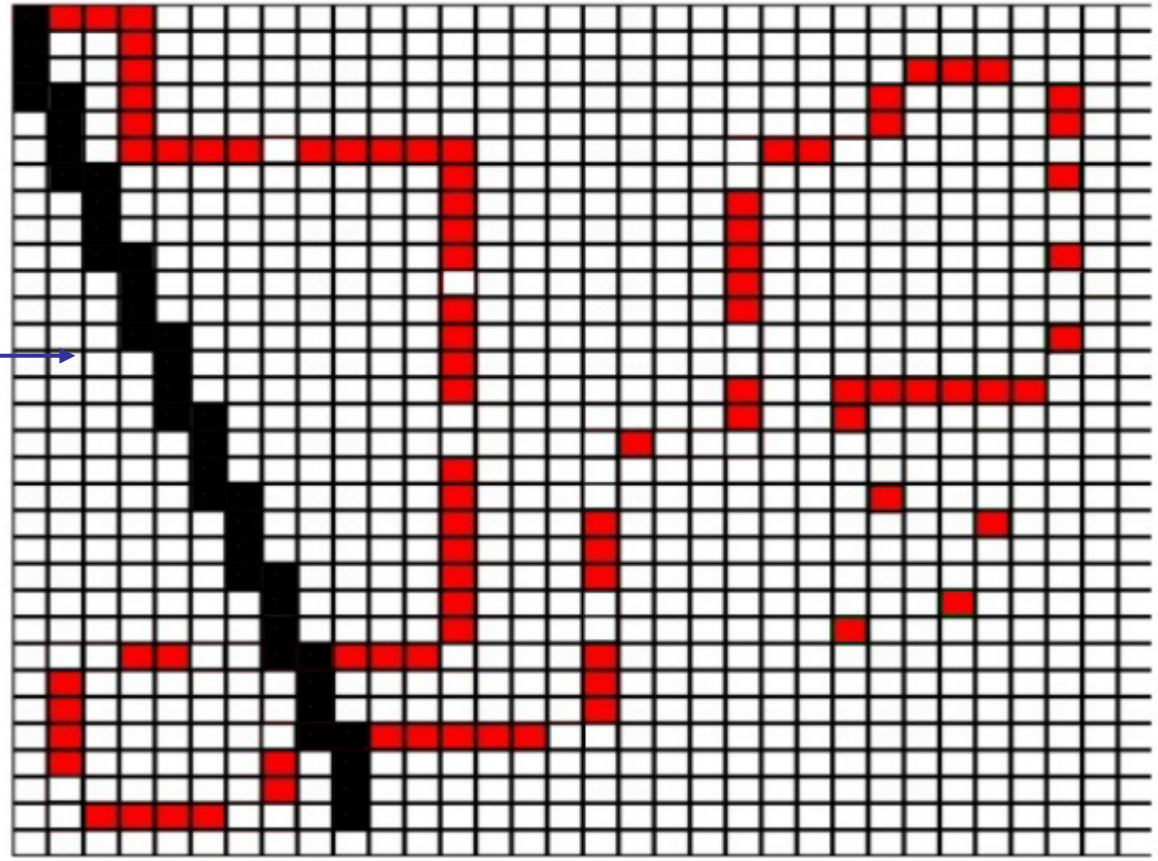


Expresión codificada en el árbol

(PROGN3 (DERECHA)
(PROGN2 (AVANCE) (IZQ))
(PROGN2 (AVANCE) (AVANCE)))

Ejemplo evaluación

Resultado de ejecutar la estrategia codificada en el individuo en este tablero



```
(PROGN3 (DERECHA)
        (PROGN2 (AVANCE) (AVANCE))
        (PROGN2 (IZQ) (AVANCE)))
```

El fitness sería alguna medida del éxito de esa estrategia en su contexto

Elementos a identificar en PG

- Identificar el conjunto de terminales :

  Ej: DERECHA, IZQUIERDA, AVANCE

- Identificar el conjunto de funciones y su aridad

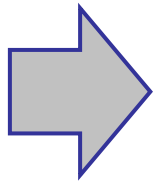
  Ej: PROGN2(_ , _) PROGN3 (_ ,_ ,_)

- Identificar la función de adaptacióno fitness
- Identificar los parámetros del algoritmo:
  - Tamaño de población
  - Número de generaciones, prob cruce y mutación
  - Fijar Profundidad de los árboles o longitud máxima de los programas….

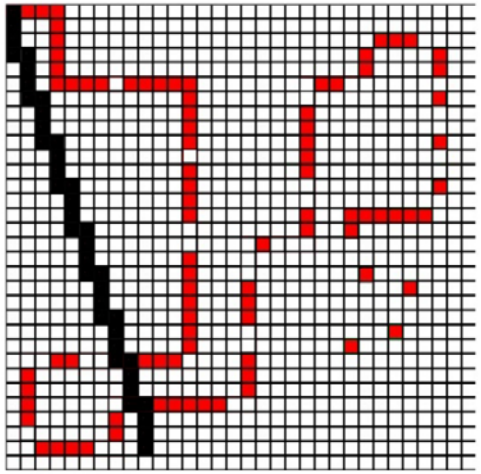
Fitness-aptitud de un individuo

- La función de aptitud, adaptación o fitness normalmente consiste en evaluar la "calidad" de un programa o individuo respecto a la resolución del problema.
- Por ejemplo, si el problema consiste en recorrer el número máximo de casillas de un tablero, valoraríamos el **número de casillas recorridas** al ejecutar el programa en cuestión.
- En el caso de la hormiga: número de trozos de comida conseguidos

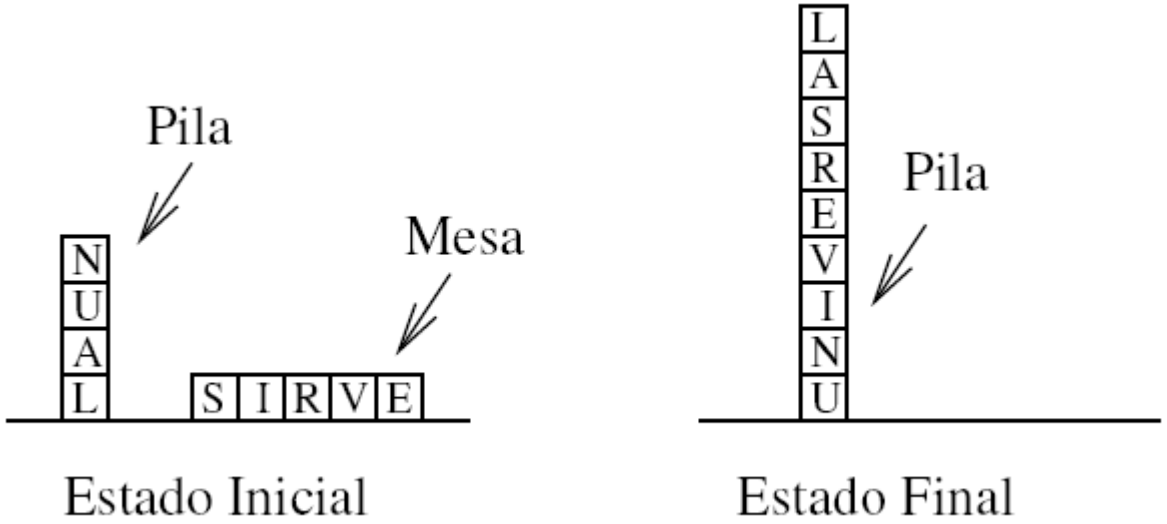
```
(PROGN3 (DERECHA)
        (PROGN2 (AVANCE) (AVANCE))
        (PROGN2 (IZQ) (AVANCE)))
```

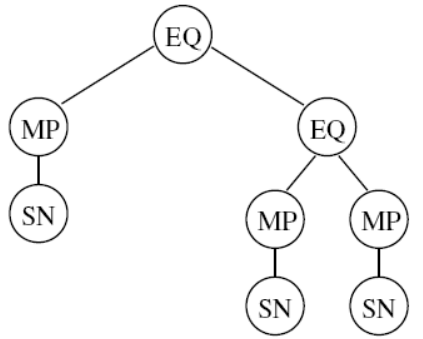


Fitness: 3



Ejemplo: Apilamiento de bloques (Koza)





(EQ(MP SN) (EQ (MP SN) (MP SN)))

- El objetivo es encontrar un programa que a partir de una configuración inicial de bloques (algunos en la mesa, algunos apilados) los coloque en el orden correcto en la pila.

  "Mover el siguiente bloque necesario a la pila hasta que no se necesiten más bloques":

  (DU (MP SN) (NOT SN))

Ejemplo: regresión simbólica

- Obtención de expresiones a partir de datos masivos
- Disponemos de datos sobre planetas: su distancia al sol (A) y su periodo orbital (P)

	A	P
Venus	0.72	0.61
Tierra	1.0	1.0
Marte	1.52	1.84
Jupiter	5.20	11.9
Saturno	9.53	29.4
Urano	19.1	83.5

- Se trata de calcular la expresión que calcula el valor de P, dada su distancia media al sol A
- Expresión a calcular:

```
P =  sqrt(A*A*A)
```

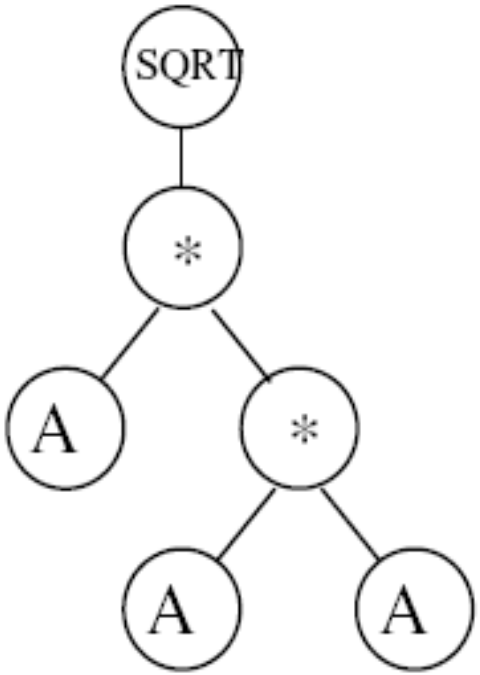
Ejemplo 2: regresión simbólica

- Identificamos terminales:

  A

- Identificamos posibles funciones:

  + - * / sqrt



Ejemplo 2: regresión simbólica

- Para generar los árboles en cada nodo se decide con cierta probabilidad si será una función o un terminal.
- Se calcula la adaptación de cada programa de la población, probándolo sobre los datos de entrada (un conjunto de medidas experimentales de P y A).
- Ejemplos de programas:

```
(+A(*(sqrt A) A))
(*A(-(*A A) (sqrt A)))
(*A(-(* sqrt A A) (+ A A)))
 . . .
```

Ejemplo 2: regresión simbólica

- Para analizar la adaptación de cada uno de los programas generados calculamos su adaptación sobre un conjunto de datos de entrada de P y A como medidas experimentales;
- La adaptación de un programa puede ser el **número de casos de prueba en los que produce un resultado correcto** o con un error muy pequeño.

	A	P
Venus	0.72	0.61
Tierra	1.0	1.0
Marte	1.52	1.84
Jupiter	5.20	11.9
Saturno	9.53	29.4
Urano	19.1	83.5

Ejemplo 3: Identificación de funciones

- Objetivo: encontrar la expresión que nos permite encontrar una solución para ecuaciones del tipo

$$ax^2 + bx + c = 0$$

- Como ya sabemos, la siguiente expresión nos permite obtener dichas soluciones

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

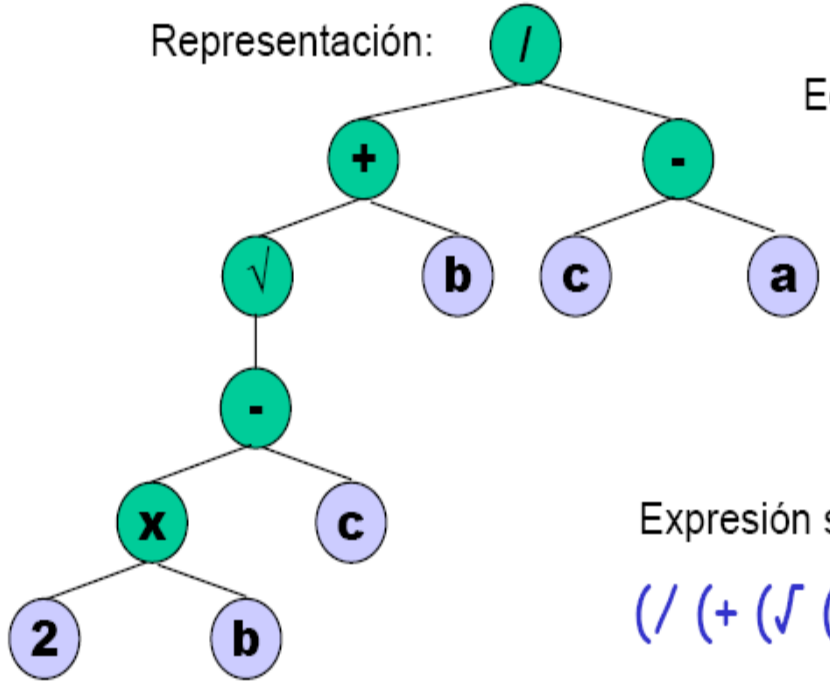
Funciones: **+** **-** **x** **/** **√**

Terminales: **a** **b** **c** **2**

Ejemplo 3: Identificación de funciones

- Representamos cada individuo o solución de la ecuación mediante un árbol. Por ejemplo:

Representación:



Ecuación: $x = \frac{\sqrt{2b-c}+b}{c-a}$

Expresión simbólica en LISP:

(/ (+ (√ (- (x 2 b) c)) b) (- c a))

Ejemplo 3: Identificación de funciones

- Dado un conjunto de ecuaciones de segundo grado se calcula cómo resuelve el individuo dicho conjunto
- Cada ecuación del conjunto de ejemplos contendrá un valor para las variable a, b y c
- Se sustituyen dichos valores en la ecuación representada por el individuo a evaluar, y así se obtiene un valor para la x
- Se sustituye dicho valor en la ecuación de segundo grado, y se comprueba si el resultado es cero
- Si es así, el individuo resuelve la ecuación, si no es así, la diferencia se utilizará para el cálculo de la aptitud del individuo

Ejemplo 3: Identificación de funciones

- Por ejemplo, dada una ecuación (de un conjunto de ecuaciones de “entrenamiento”)

$$-4x^2+4x-1=0$$

- Sustituyendo estos valores en el individuo ejemplo anterior se obtiene:

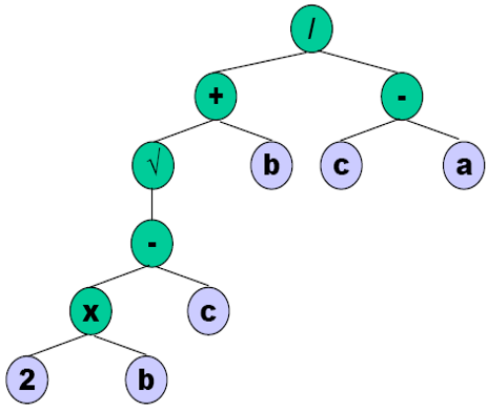
$$x=\frac{\sqrt{2b-c}+b}{c-a}=\frac{\sqrt{8-(-1)}+4}{(-1)-(-4)}=\frac{\sqrt{9}+4}{3}=\frac{7}{3}=2.33$$

- Sustituyendo x por su valor 2.33 en la ecuación de segundo grado se obtiene

$$-4x^2+4x-1=$$
$$-4(2.33)^2+4(2.33)-1=$$
$$-21.72+9.32-1=-13.4$$

Representación individuo

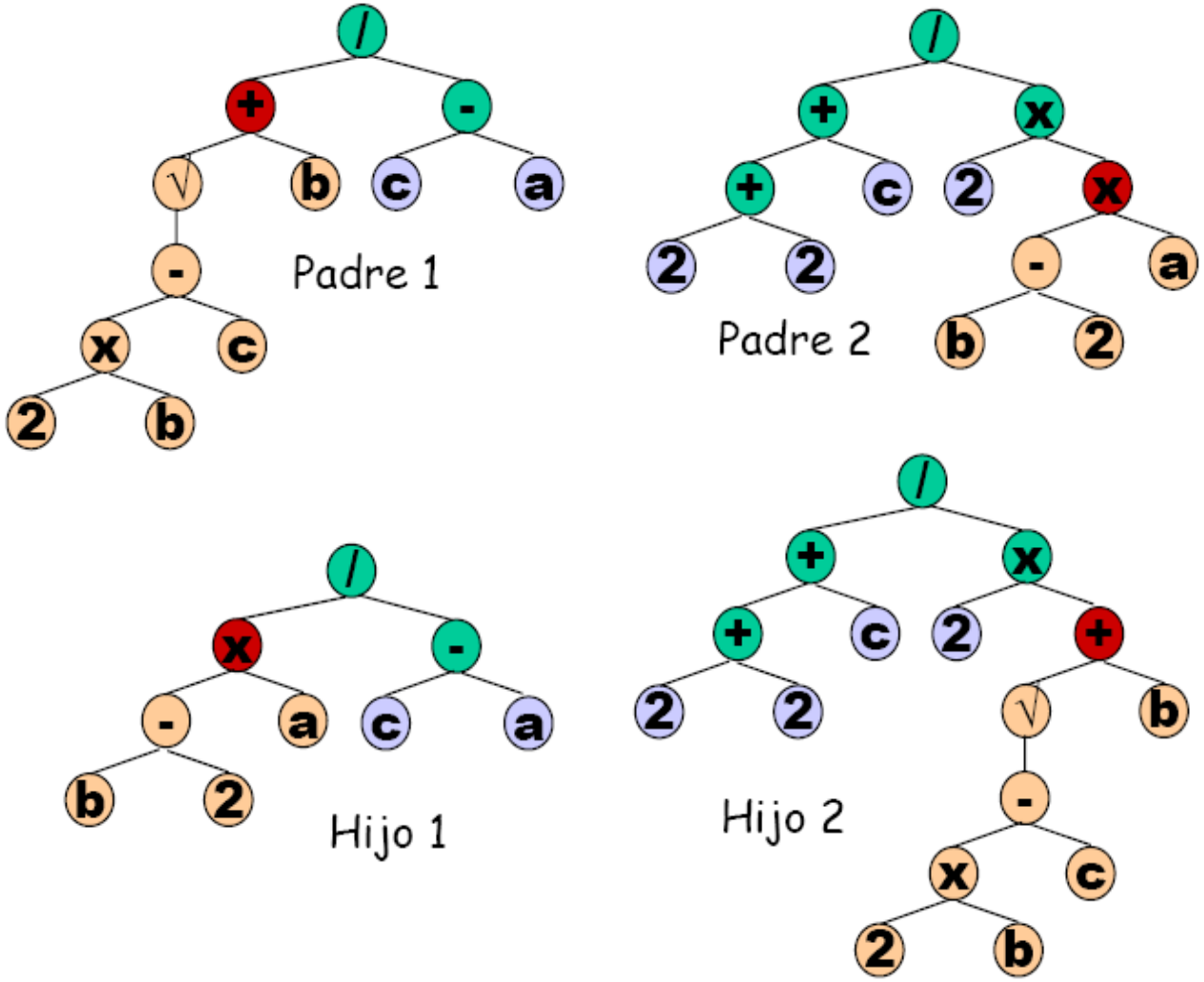
```
public class Cromosoma {
//el genotipo será algo como un árbol
 private double aptitud;
 private Arbol arbol;
 private int hMaxima;
 private final String[] cjtoTerms={"a","b","c","2","0"};
 private final String[] cjtoFuns={"+","-","*","/","sqrt"};
  . . .

}
```



(/ (+ (√ (- (x 2 b) c)) b) (- c a))

Operador de Cruce

- Intercambiamos los subárboles
- Los puntos de intercambio son los nodos rojos
- Una estrategia consiste en seleccionar los nodos internos (funciones) con una probabilidad alta (0.9) y cualquier nodo con el resto de la probabilidad



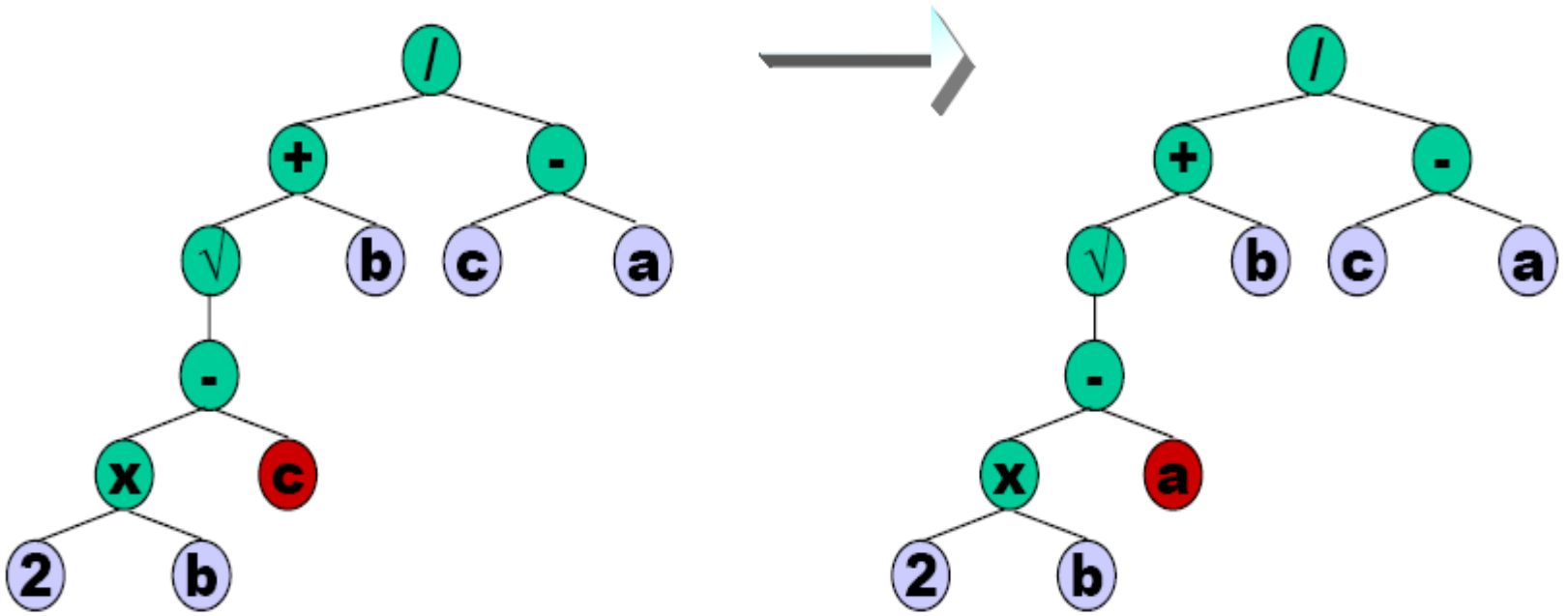
Operador de mutación

La mutación consiste en generar un nuevo programa a partir de un único progenitor. Las técnicas de mutación más comunes son

- Mutación terminal
- Mutación funcional
- Mutación de árbol
- Mutación de permutación
- Subárbol
- Hoist
- Contracción
- Expansión

Mutación terminal

- Mutación terminal : Se selecciona al azar un símbolo terminal dentro del individuo, y se sustituye por otro diferente del conjunto de símbolos terminales posibles

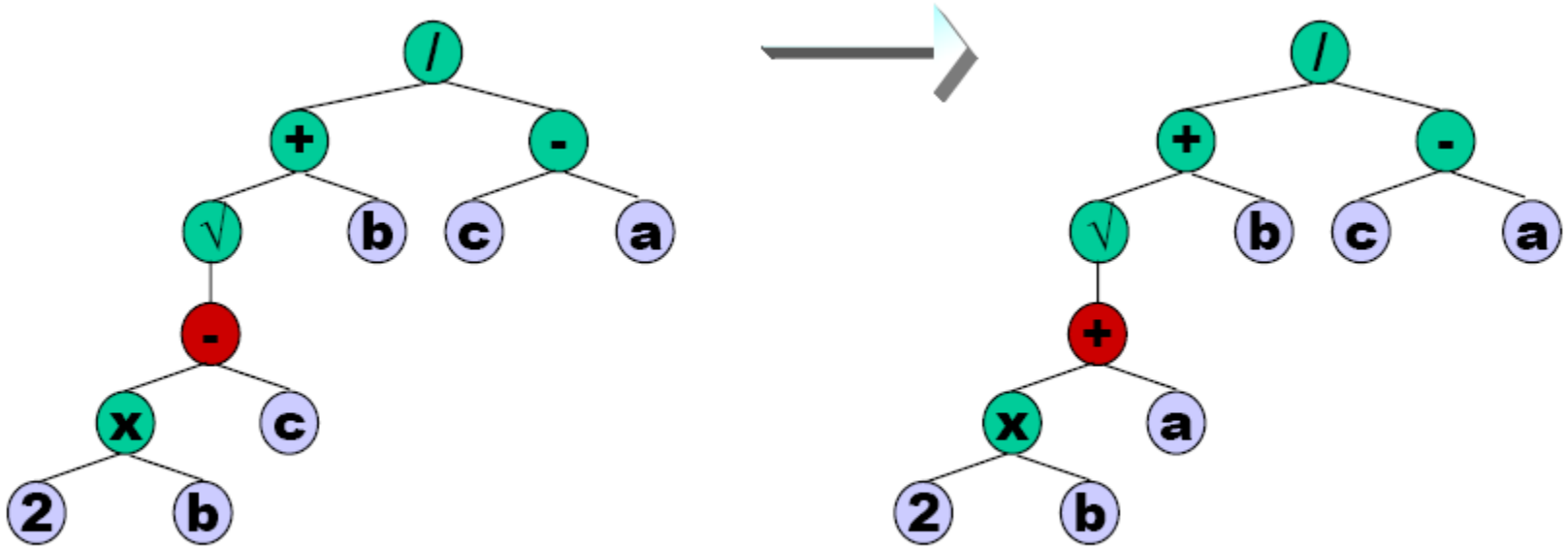


Mutación terminal

```
public void mutacionTerminalSimple(){
 char []terminales = new char[4];
 terminales[0] = 'a';terminales[1] = 'b';
 terminales[2] = 'c';terminales[3] = '2';

for (int i = 0; i < this.Poblacion; i++) {
 Cromosoma c = (Cromosoma) individuos.get(i);
 double numAle = Math.random();
 if(numAle < this.PrMut){
   int numAle2 = (int) (Math.random()*4);
   Simbolo s = getTerminalAleatorio(c);
   s.setTerminal(terminales[numAle2]);
  }
 }
}
```

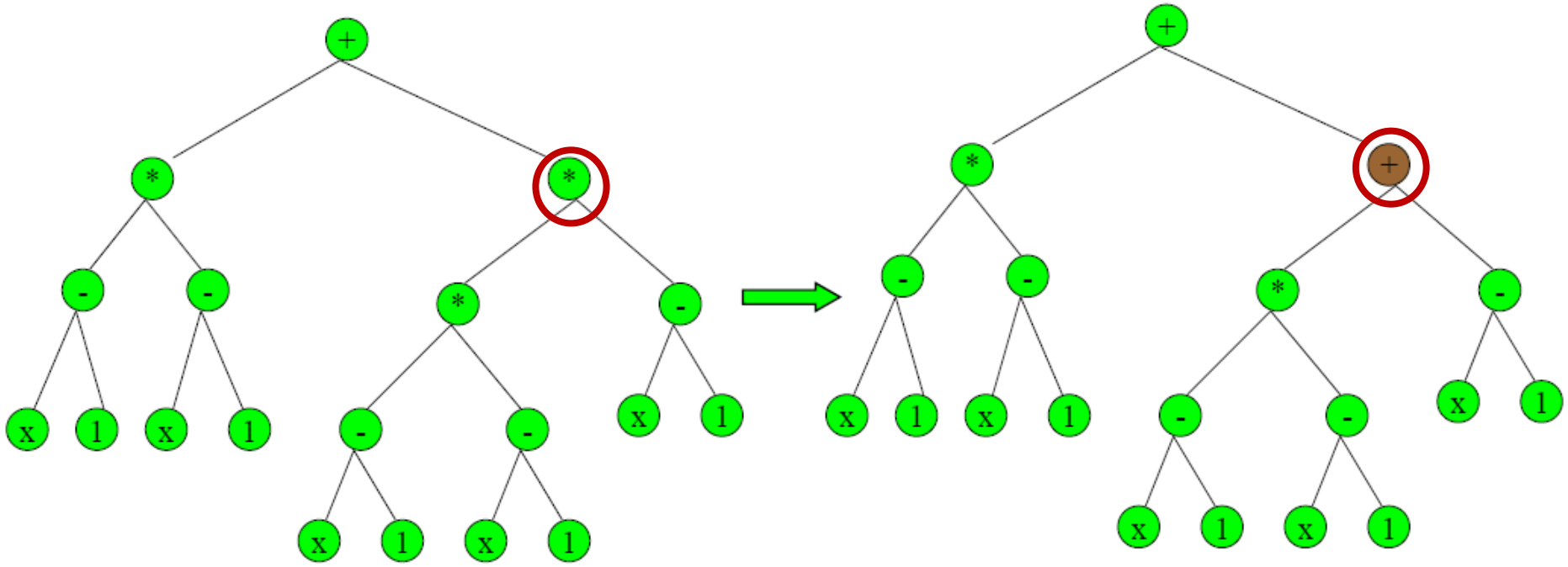
Mutación funcional

- Mutación funcional : Se selecciona al azar una función dentro del individuo, y se sustituye por otra diferente del conjunto de funciones posibles con el mismo número de operandos



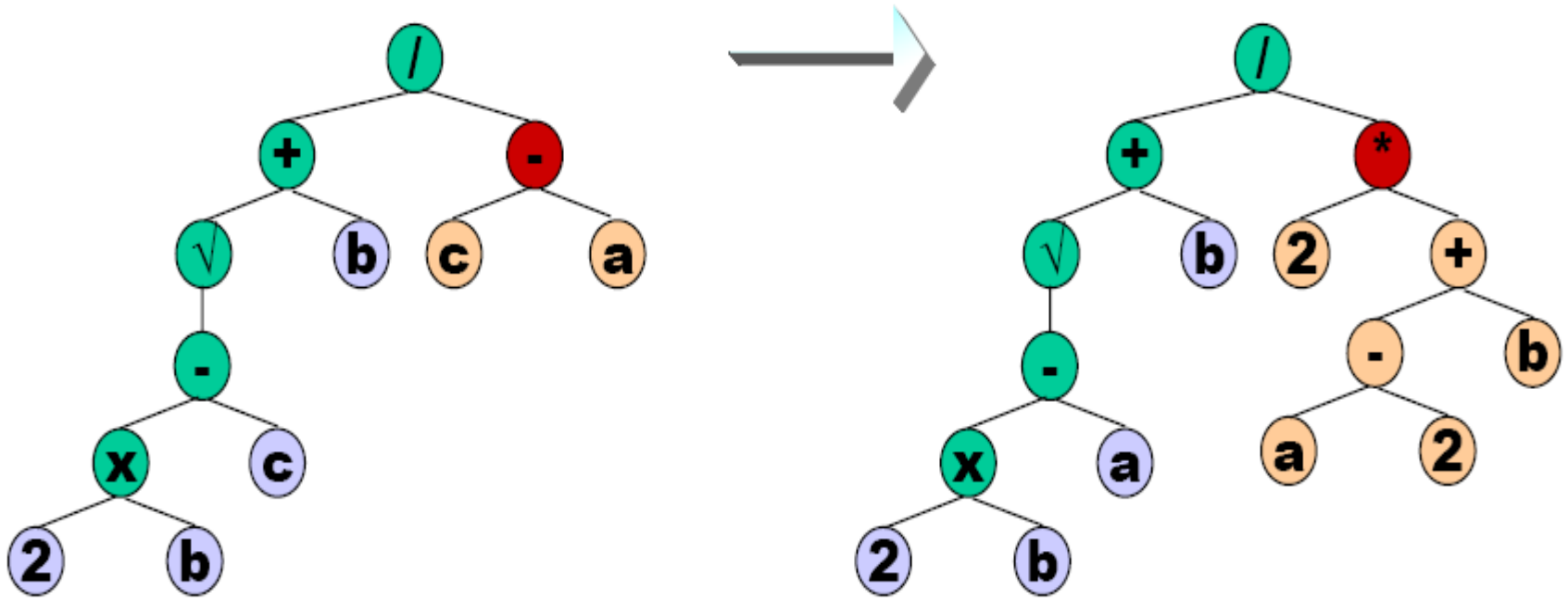
Ejemplo mutación Funcional

Cambia un nodo función o terminal por otro válido



Mutación de árbol-subárbol

- Mutación de árbol : Se selecciona un subárbol del individuo, se elimina totalmente el subárbol seleccionado y en su lugar se incorpora un nuevo subárbol generado aleatoriamente

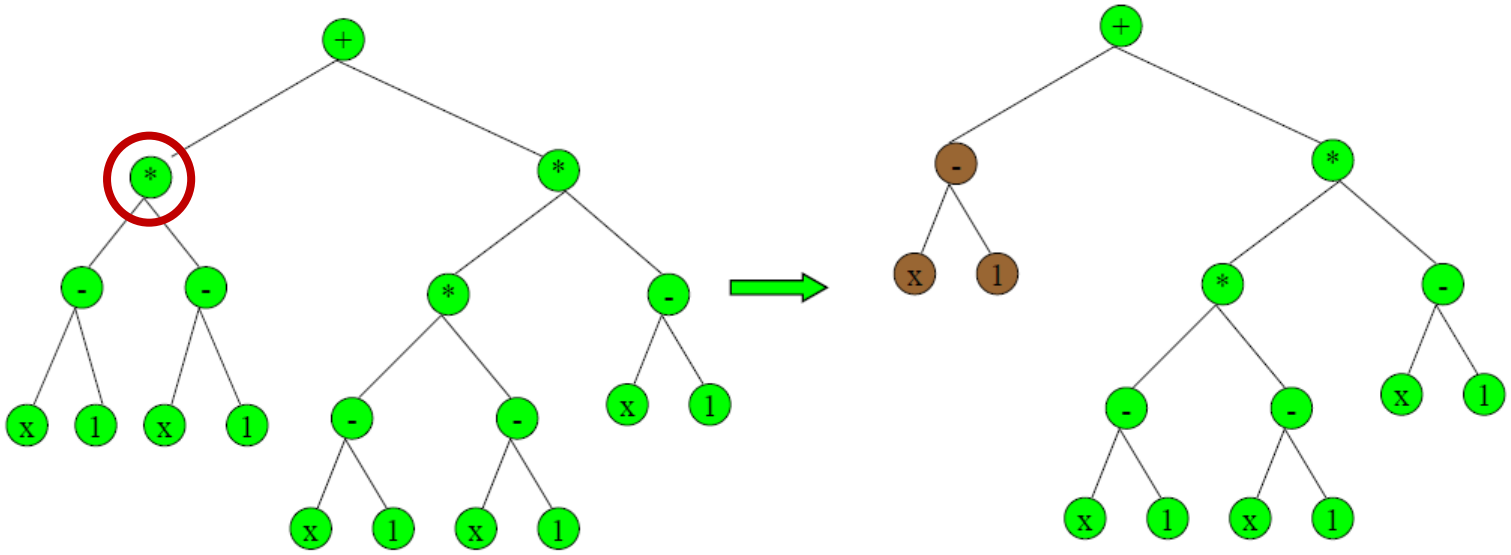


Ejemplo mutación de árbol

```
void TIndividuo::muta(float probMutacion){

 //Ejemplo de mutación en la que los individuos a
 //mutar son directamente reconstruidos.

    //Comprobamos si vamos a mutar al individuo
    if(rand()/(RAND_MAX+1.0f) < probMutacion ){

        borraArbol(this->arbol);
        this->aptitud= 0.0f;
        this->ajustado=false;
        this->puntuacion = 0.0f;
        this->inicializa();

    }

}
```

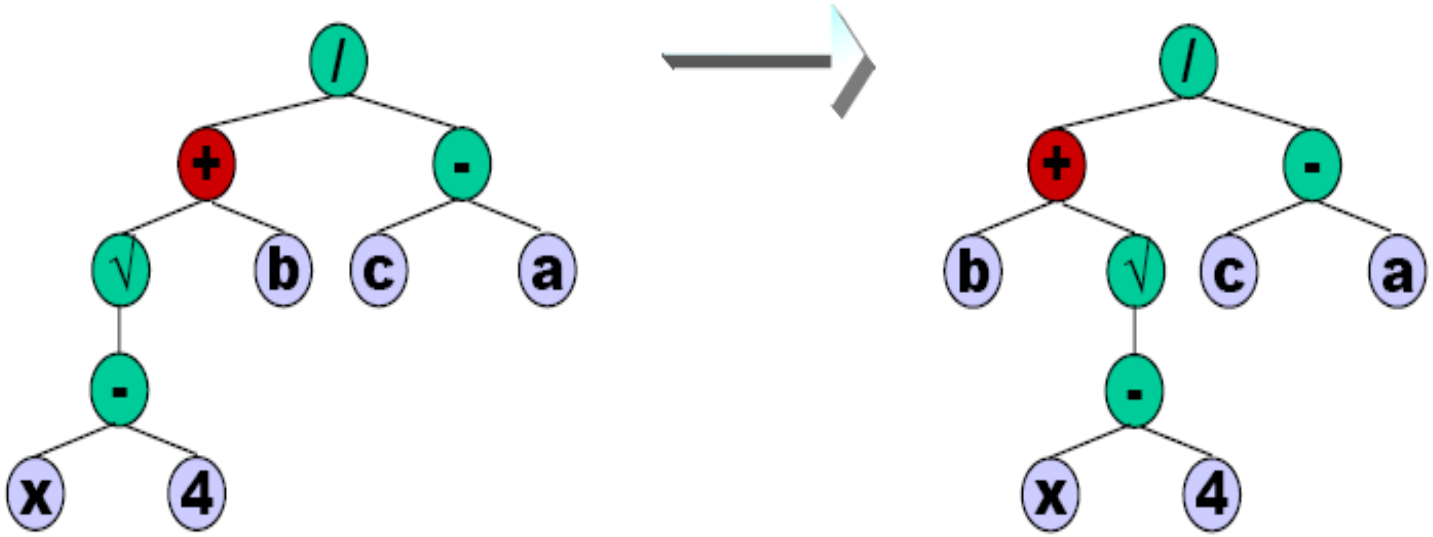
Ejemplo mutación subárbol

Regenera un subárbol



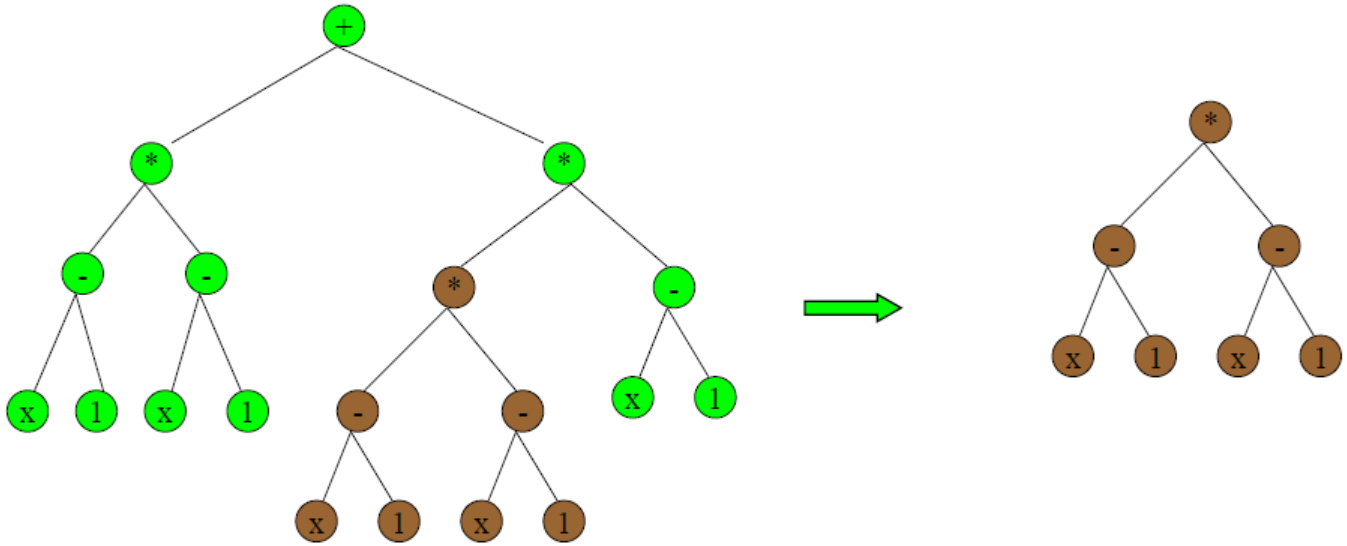
Mutación Permutación

- Intercambia el orden de la lista de argumentos de una función



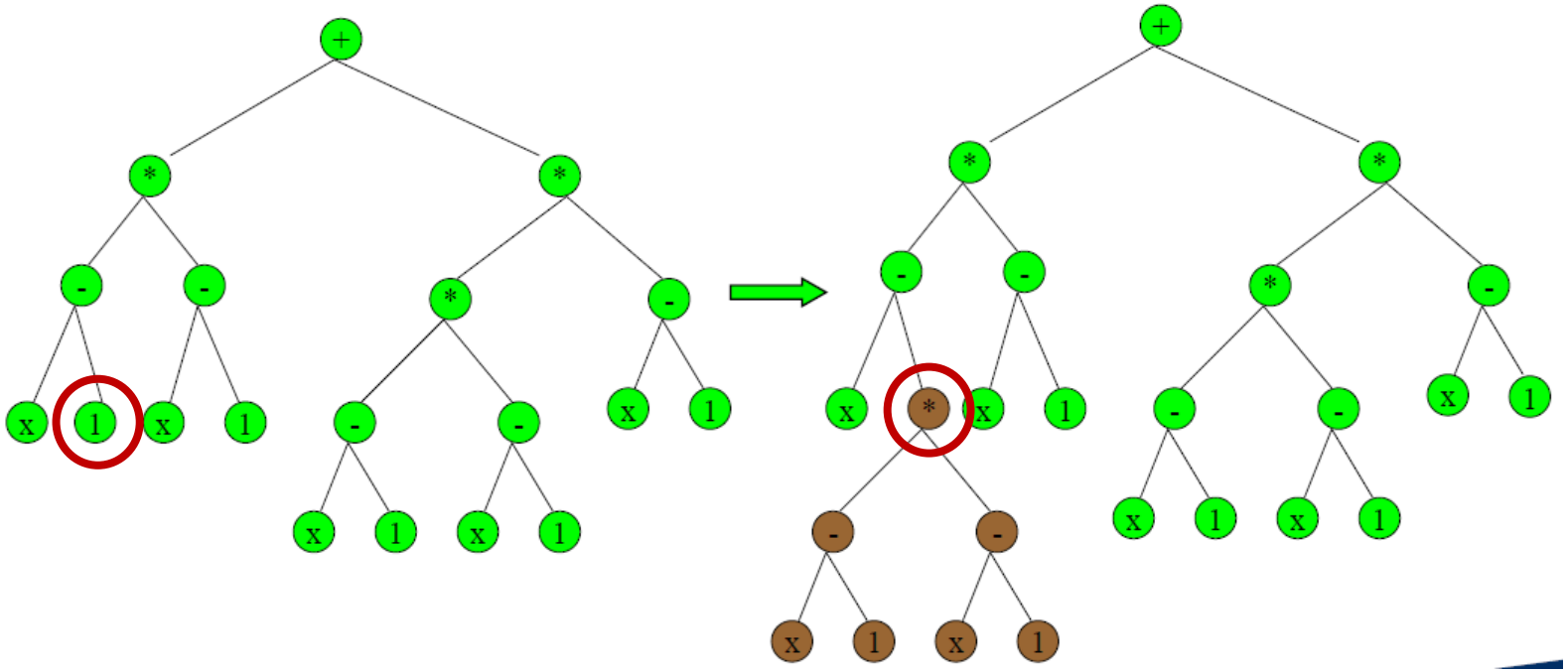
Mutación hoist

Muta a un subárbol



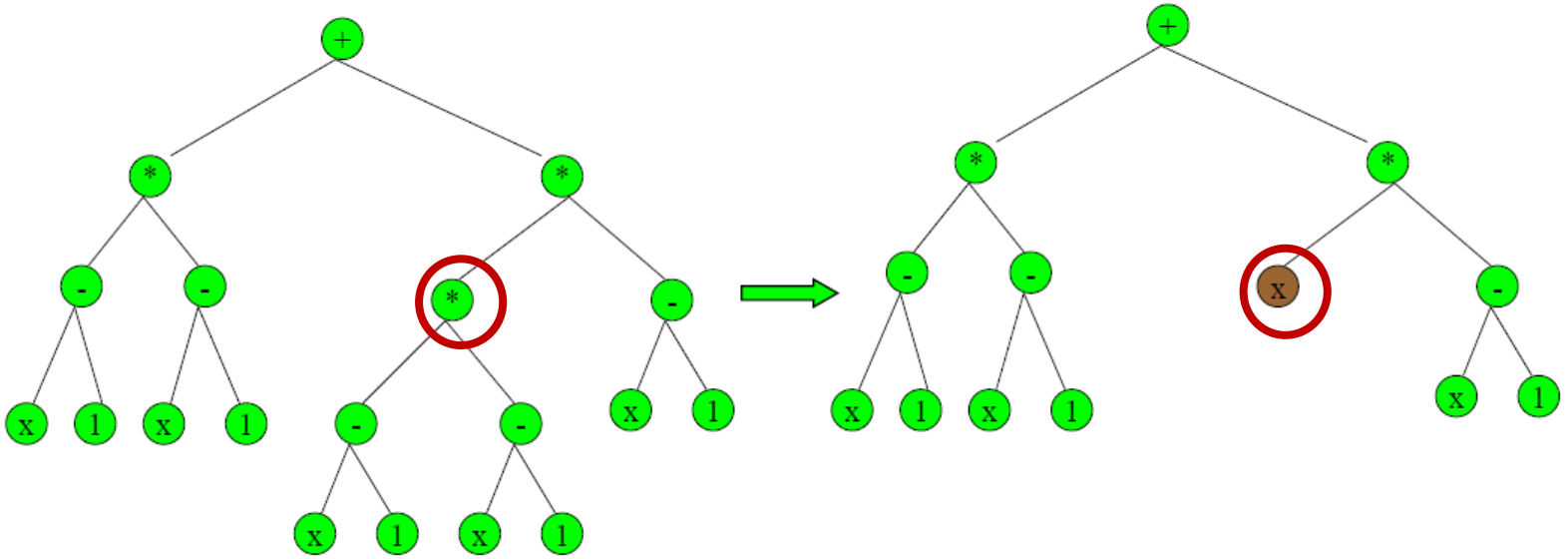
Mutación expansión

Expande un nodo terminal



Mutación: contracción

Reduce un subárbol a un terminal



Creación de los individuos:

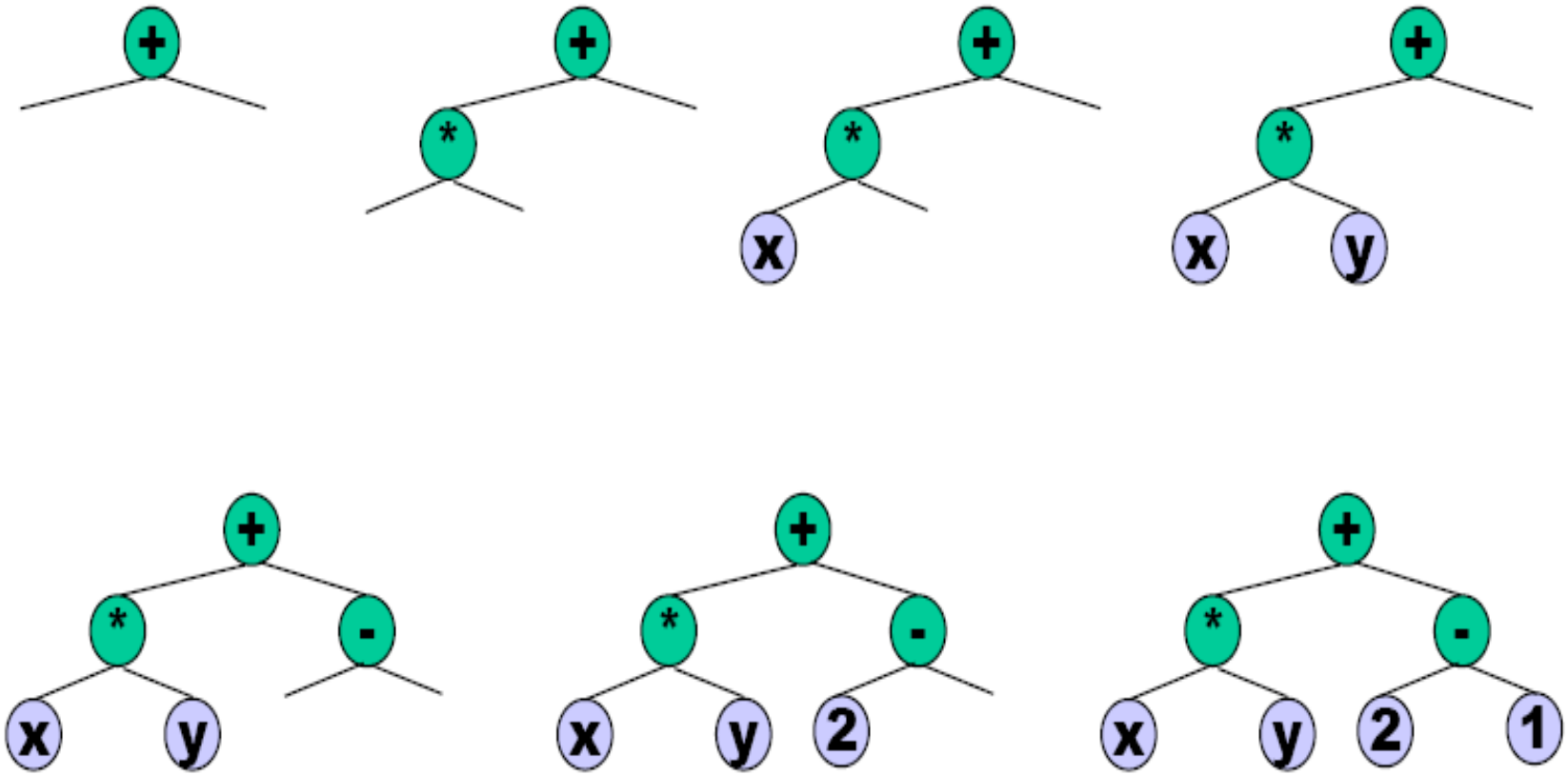
- Técnicas de inicialización
  - Inicialización completa (*Full initialization*)
  - Inicialización creciente (*Grow initialization*)
  - Inicialización Ramped & Half

Inicialización completa: (*Full initialization*)

- Vamos tomando nodos del conjunto de funciones hasta llegar a una máxima profundidad del árbol definida previamente
- Una vez llegados a la profundidad máxima los símbolos sólo se toman del conjunto de **símbolos terminales**

```
funcion inicializacionCompleta(profundidad) {
 si profundidad < maximaProfundidad entonces
   nodo ← aleatorio(conjFunciones)
   para i = 1 hasta número de hijos del nodo hacer
      Hijo_i = inicializacionCompleta(profundidad+1 )
 eoc
   nodo ← aleatorio(conjTerminales)
 devolver nodo
}
```

Inicialización completa

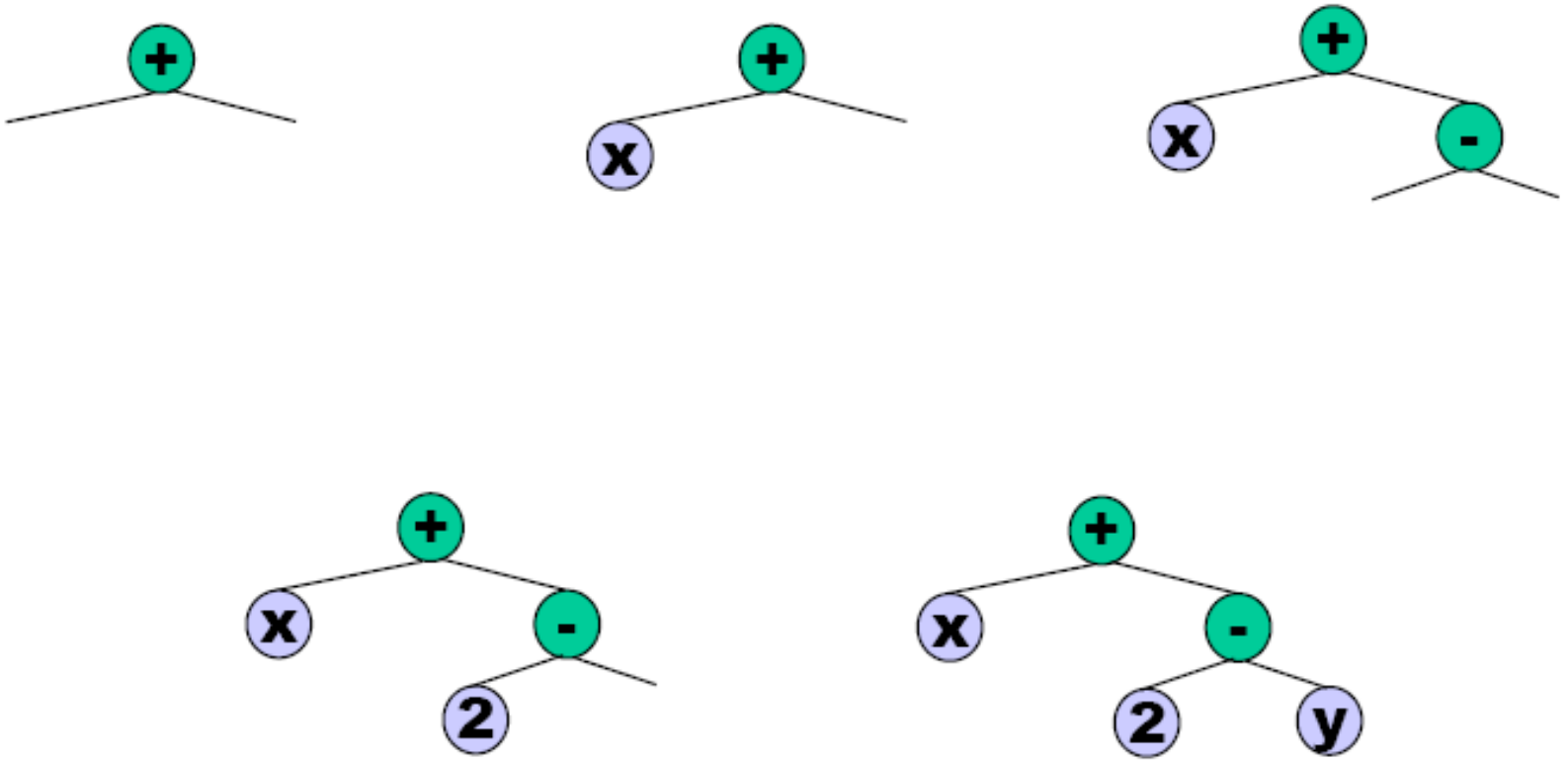


Inicialización creciente: (Grow *initialization*)

- Vamos tomando nodos del conjunto completo (funciones y terminales) hasta llegar al límite de profundidad especificado previamente
- Una vez llegados a la profundidad máxima este método de inicialización se comporta igual que el método de inicialización completa

```
función inicializacionCreciente(profundidad) {
  si profundidad < maximaProdundidad árbol entonces
   nodo ← aleatorio(conjFunciones ⊕ conjTerminales)
   para i = 1 hasta número de hijos del nodo hacer
      Hijo_i = inicializacionCreciente(profundidad+1 )
 eoc
   nodo ← aleatorio(conjTerminales)
 devolver nodo
}
```

Inicialización creciente



Inicialización Ramped and half

- Asegura diversidad: Combina los métodos creciente y completo.
- Dada una profundidad máxima del árbol **D** el método Ramped and Half divide el tamaño de la población en **D -1** grupos.
- Cada grupo utiliza una profundidad diferente máxima del árbol (2 , ..., D) , con lo que se crea la mitad de cada grupo mediante el método creciente y la otra mitad utilizando el método completo .
- El resultado es una mezcla de árboles irregulares de diferentes profundidades creadas por el método creciente y árboles más regulares creados por el método completo

Inicialización Ramped and half

- Ejemplo 90 individuos, profundidad máxima D=4
- Se hacen D-1 grupos (3) de 30 individuos cada uno:
  - Grupo 1: árboles profundidad **2**:
    - 15 completa ,15 creciente
  - Grupo 2: árboles profundidad **3**:
    - 15 completa ,15 creciente
  - Grupo 3: árboles profundidad **4**:
    - 15 completa ,15 creciente

Aptitud o fitness de un individuo

- El valor de aptitud asignado a un programa puede medirse de formas diferentes. Por ejemplo
  - La cantidad de error entre sus salidas y las salidas deseadas
  - La cantidad de tiempo o esfuerzo necesario para llevar un sistema a un determinado estado objetivo
  - La cantidad de puntos o elementos obtenidos con la estrategia que codifica el árbol del individuo

Problemas en PG: **Bloating**

- Bloating: crecimiento sin control de los árboles durante el proceso evolutivo

- Control del bloating
  - Modificar los operadores para que no construyan individuos muy grandes
  - Incluir algún término en la función de evaluación **que penalice a los árboles o expresiones según su tamaño**

Control del Bloating

- A medida que avanzan las generaciones, el tamaño de los individuos crece (sin necesidad de que también mejore el fitness)
- Crecimiento prácticamente exponencial
- Puede estar causado por “intrones” (código inútil).

```
a + (a-a+a-a)

0*(a*b+c*d)

If (False) then {...}
```

Control del Bloating

- Penalizar a los individuos grandes en la función de fitness (suponiendo minimización):

  f'(x) = f(x) + k * nodos-en(x)

- Limitar el tamaño o profundidad máxima
- Hacer torneos con fitness y si hay empate elegir a los más pequeños
- Evitar el cruce destructivo (que se produzcan hijos peores que los padres):
  - Ejecutar la operación de cruce N-veces y quedarse con los mejores hijos

Control Bloating 1: método **Tarpeian**

```
IF size(program) > average_pop_size AND random_int MOD n = 0
THEN
     return( very_low_fitness );
ELSE
     return( fitness(program) );
```

- Elimina aquellos programas de longitud mayor que la media, de vez en cuando (con probabilidad 1/n) (n=2, la mitad de los programas grandes muere)
- Si se crea un programa mayor que la media, tiene grandes posibilidades de morir
- Si sobrevive y tiene fitness mejor que la media, se reproducirá, y el tamaño medio de la población se incrementará

Control Bloating 2: penalización bien fundamentada.

- Penalización bien fundamentada (Poli and McPhee, 2008b)

  f'(x) = f(x) + k * nodos-en(x)

  k_t = Covarianza(l,f) / Varianza(l)

  l = tamaño del individuo

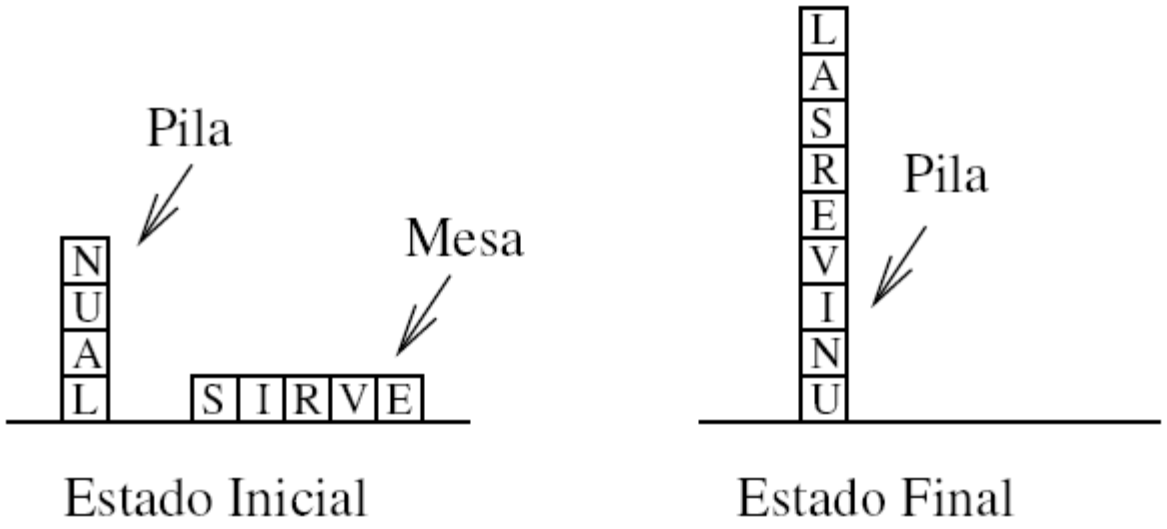
  f = fitness del individuo

- Hay que recalcular k en cada generación (por eso depende del tiempo)
- Con esto se consigue que el tamaño medio de los individuos en todas las generaciones sea aproximadamente el mismo que el tamaño medio en la generación cero => No hay bloat

Ejemplos

Ejemplo: Apilamiento de bloques (Koza)

- El objetivo es encontrar un programa que a partir de una configuración inicial de bloques (algunos en la mesa, algunos apilados) los coloque en el orden correcto en la pila.
- En el ejemplo, el orden correcto es el que corresponde al deletreo de la palabra UNIVERSAL.

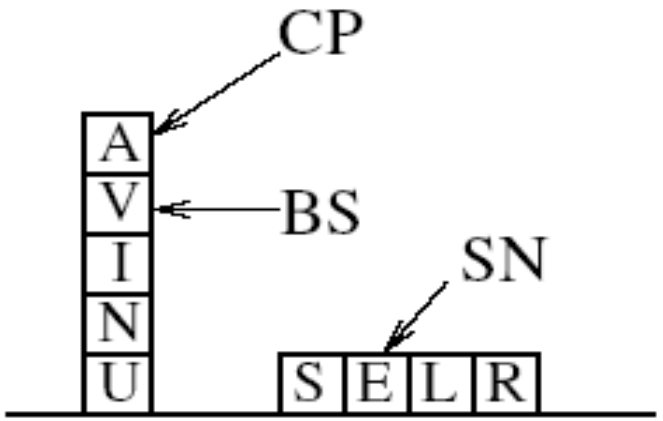


Ejemplo: Apilamiento de bloques (Koza)

- Los terminales para este problema son un conjunto de sensores T={CP,BS,SN} que devuelven:
  - CP (cima pila): nombre del bloque de la cima de la pila. Si la pila está vacía devuelve NIL.
  - BS (bloque superior correcto): nombre del bloque más alto en la pila tal que él y todos los bloques que tiene por debajo están en el orden correcto. Si no existe tal bloque BS devuelve NIL.

Ejemplo: Apilamiento de bloques (Koza)

- SN (siguiente necesario): nombre del bloque que se necesita poner inmediatamente sobre BS en el objetivo UNIVERSAL (independientemente de que haya o no bloques incorrectos en la pila). Si no se necesitan más bloques devuelve NIL.



- Estos terminales toman como valores las etiquetas de los bloques o NIL y representan las variables del programa.

Ejemplo: Apilamiento de bloques (Koza)

También se dispone de cinco funciones:

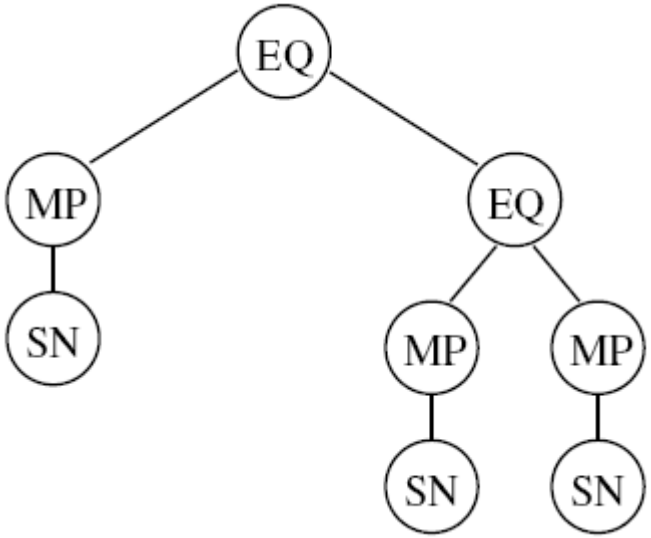
- MP <x>(mover a pila) pone el bloque x en la cima de la pila si x está sobre la mesa. (En Lisp cada función devuelve un valor, que a menudo se ignora).
- MM <x>(.Mover a mesa.) mueve el bloque de la cima de la pila a la mesa si el bloque x está en cualquier punto de la pila.
- DU(acción, predicado) (“do until”) ejecuta la acción hasta que predicado se hace cierto (TRUE).
- NOT(expresión1) devuelve cierto (TRUE) si expresión1 es NIL; en otro caso devuelve NIL,
- EQ(expresión1, expresión2) devuelve cierto (TRUE) si expresión1 y expresión2 son iguales.

Ejemplo: Apilamiento de bloques (Koza)

- El algoritmo trabaja con árboles de análisis que representan programas construidos con los terminales y funciones descritos. El programa:

  (EQ(MP SN) (EQ (MP SN) (MP SN)))

- Mueve el siguiente bloque que se necesite a la pila tres veces y está representado por el árbol:



Ejemplo: Apilamiento de bloques (Koza)

- Se genera una población inicial de programas aleatorios (árboles sintácticamente correctos y con un límite de profundidad) usando las funciones y terminales definidos.
- La adaptación de cada programa se calcula como el resultado de su ejecución sobre distintas configuraciones iniciales.
- La adaptación de cada programa es una función del número de casos de prueba en los que se produce el resultado correcto.

Datos leídos de fichero →

```
* U N I V E R S A L *
U * N I V E R S A L *
U N * I V E R S A L *
U N I * V E R S A L *
U N I V * E R S A L *
U N I V E * R S A L *
U N I V E R * S A L *
. . .
```

Hasta el asterisco son bloques sobre la mesa, desde el asterisco son bloques sobre la pila.

Ejemplo: Apilamiento de bloques (Koza)

- La aptitud de un programa es el número de casos favorables (configuraciones iniciales de bloques) para los que la pila era correcta después de ejecutar el programa.
- Koza usó 166 casos de prueba diferentes, construidos cuidadosamente para recoger las distintas clases de posibles configuraciones iniciales.
- La población inicial contiene entre 50 y 300 programas generados aleatoriamente.

Ejemplo: Apilamiento de bloques (Koza)

Ejemplos:

(EQ (MM CP) SN)

- Mover la cima actual de la pila a la mesa, y ver si es igual al siguiente necesario.
- No consigue ningún progreso: aptitud 0.

(MP BS)

- Mover a la pila el bloque más alto correcto de la pila.
- No hace nada, pero permite conseguir un caso de aptitud correcta: el caso en el que todos los bloques ya estaban en la pila en el orden correcto (aptitud = 1).

Ejemplo: Apilamiento de bloques (Koza)

Ejemplos:

(EQ(MP SN) (EQ (MP SN) (MP SN)))

- Mover el siguiente bloque que se necesite a la pila tres veces.
- Hace algunos progresos y consigue 4 casos correctos (aptitud=4).
- EQ sirve únicamente como una estructura de control.
- Lisp evalúa la primera expresión, después la segunda, y después compara sus valores. EQ ejecuta las dos expresiones en secuencia, y no importa si sus valores son iguales.

Ejemplo: Apilamiento de bloques (Koza)

- En la generación 5, la población contenía muchos programas de aptitudes relativamente buenas. El mejor:

  (DU (MP SN) (NOT SN))

- Mover el siguiente bloque necesario a la pila hasta que no se necesiten más bloques.
- Funciona en todos aquellos casos en que los bloques de la pila ya están en el orden correcto.
- Hubo 10 de estos casos (aptitud=10).
- Este programa usa el bloque constructivo (MP SN) que se descubrió en la primera generación y que ha resultado útil aquí.

Ejemplo: Apilamiento de bloques (Koza)

- En la generación 10 se obtuvo un programa completamente correcto (aptitud 166):

  (EQ (DU (MM CP) (NOT CP)) (DU (MP SN) (NOT SN)))

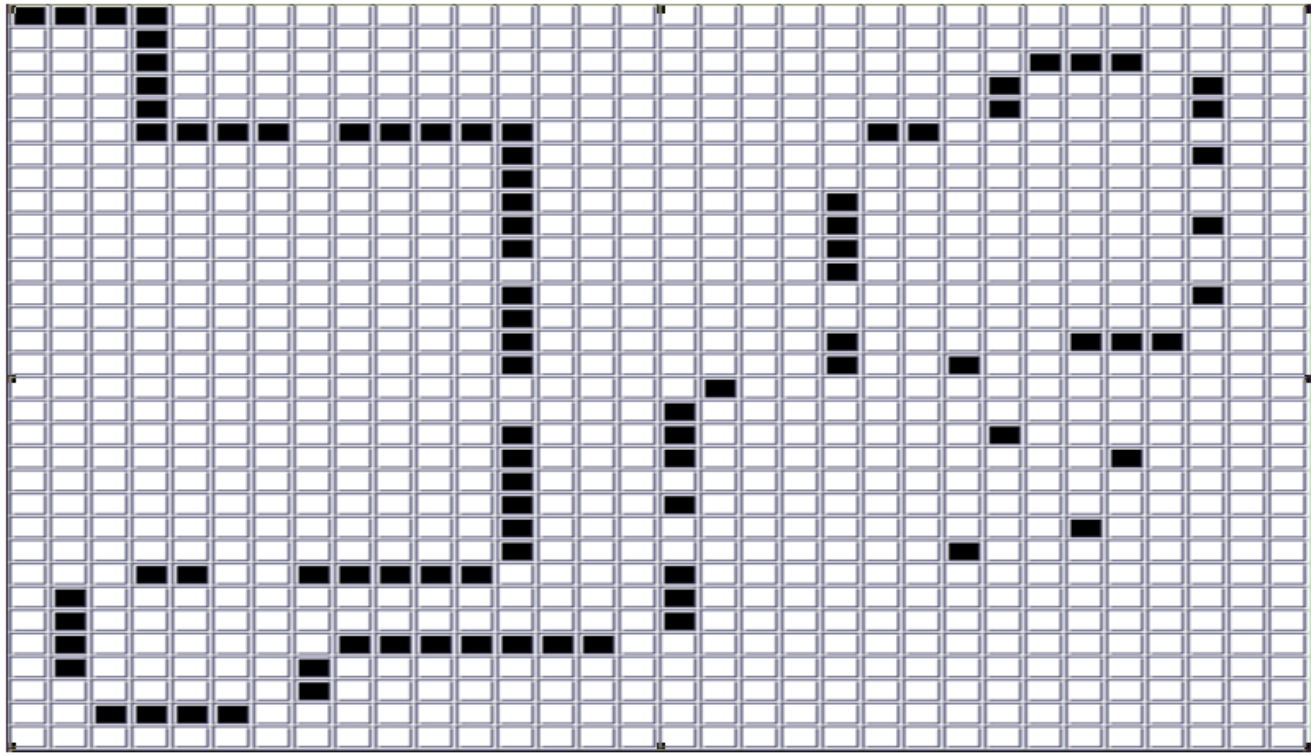
- Extensión del mejor programa de la generación 5.
- Vacía la pila sobre la mesa y después mueve el siguiente bloque que se necesita a la pila hasta que ya no se necesitan más bloques.

Ejemplo Hormiga: Rastro de Santa Fe

- Se trata de diseñar una hormiga artificial capaz de encontrar toda la comida situada a lo largo de un rastro irregular. El objetivo es utilizar programación genética para obtener el programa que realiza esta tarea.
- La hormiga artificial se mueve en un tablero (toroidal) de 32 x 32. La hormiga comienza en la esquina superior izquierda del tablero, identificada por las coordenadas (0,0), y mirando en dirección este.
- Un modelo de rastro propuesto, con el que se han realizado diversos estudios, se conoce como "rastro de Santa Fe" , y tiene una forma irregular compuesta de 89 "bocados" de comida.

Ejemplo Hormiga

- El rastro no es recto y continuo, sino que presenta huecos de una o dos posiciones, que también pueden darse en los ángulos.



Ejemplo Hormiga

Terminales = {

AVANZA

DERECHA

IZQUIERDA

}

donde

- AVANZA mueve la hormiga hacia delante en la dirección a la que mira en ese momento
- DERECHA gira la hormiga 90° a la derecha
- IZQUIERDA la gira 90° a la izquierda.

Ejemplo Hormiga

Funciones = { SIC, PROGN2, PROGN3 }

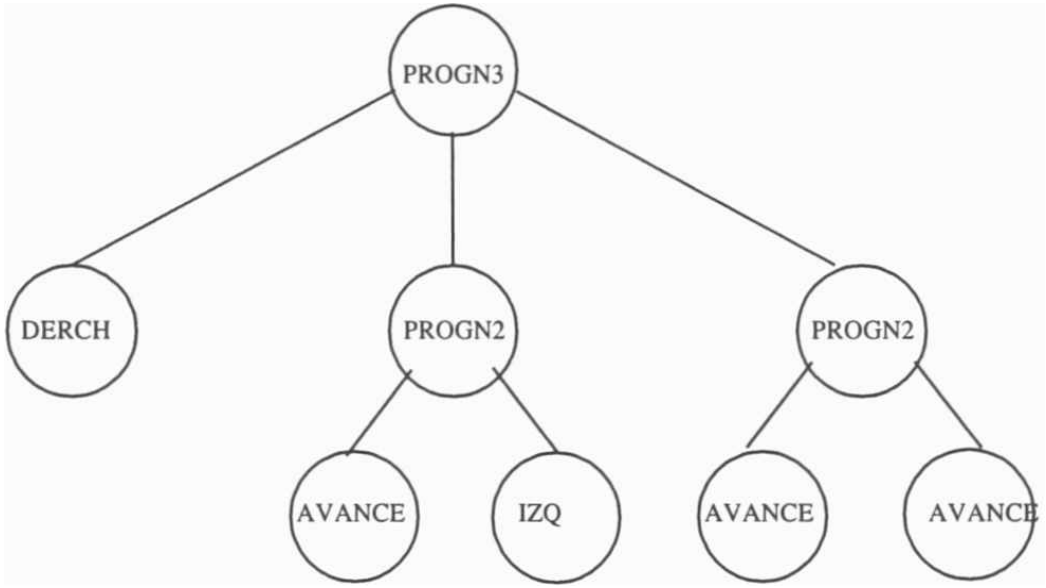
- ***SIC (a,b)*** *:* el operador SI_Comidadelante toma dos argumentos y ejecuta *a* si se detecta comida delante y *b* en otro caso.
- ***PROGN2(a,b)*** : evalúa *a*, luego *b*, y devuelve el valor de *b*.
- ***PROGN3(a,b,c)*** : evalúa *a, b* luego *c*, devolviendo el valor de *c*

Ejemplo Hormiga

(PROGN3 (DERECHA) (PROGN2 (PROGN2 (AVANZA) (IZQUIERDA)) (AVANZA) (AVANZA))

(PROGN3 (SIC) (SIC) (PROGN3 (SIC) (SIC (SIC AVANZA DERECHA) (SIC DERECHA DERECHA)) AVANZA))

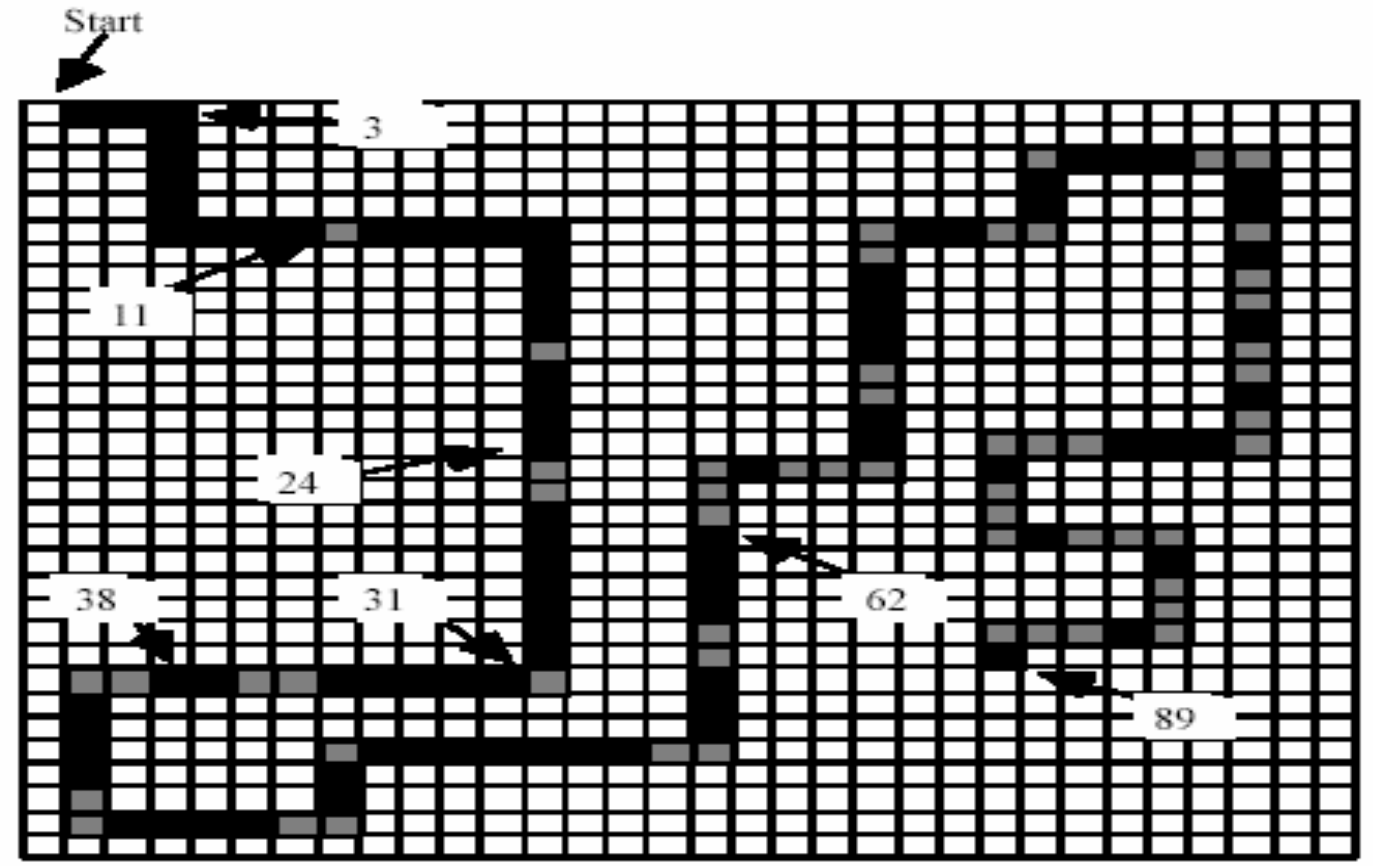
- Las expresiones de este tipo pueden representarse como árboles de análisis, compuestos de funciones y terminales.



Ejemplo Hormiga

- Fitness: cantidad de alimento comido por la hormiga dentro de un espacio de tiempo razonable al ejecutar el programa a evaluar.
- Se considera que cada operación de movimiento o giro consume una unidad de tiempo.
- En nuestra versión del problema limitaremos el tiempo a 500 pasos. El tablero se va actualizando a medida que desaparece la comida.

Ejemplo de la Hormiga



Falla AVANZA

(PROGN2 (DERECHA) (IZQUIERDA))

Ejemplo de la Hormiga

- AVANZA sin mirar o girar

```
(PROGN2 (AVANZA) (AVANZA))
```

- Contiene condicional pero no AVANZA

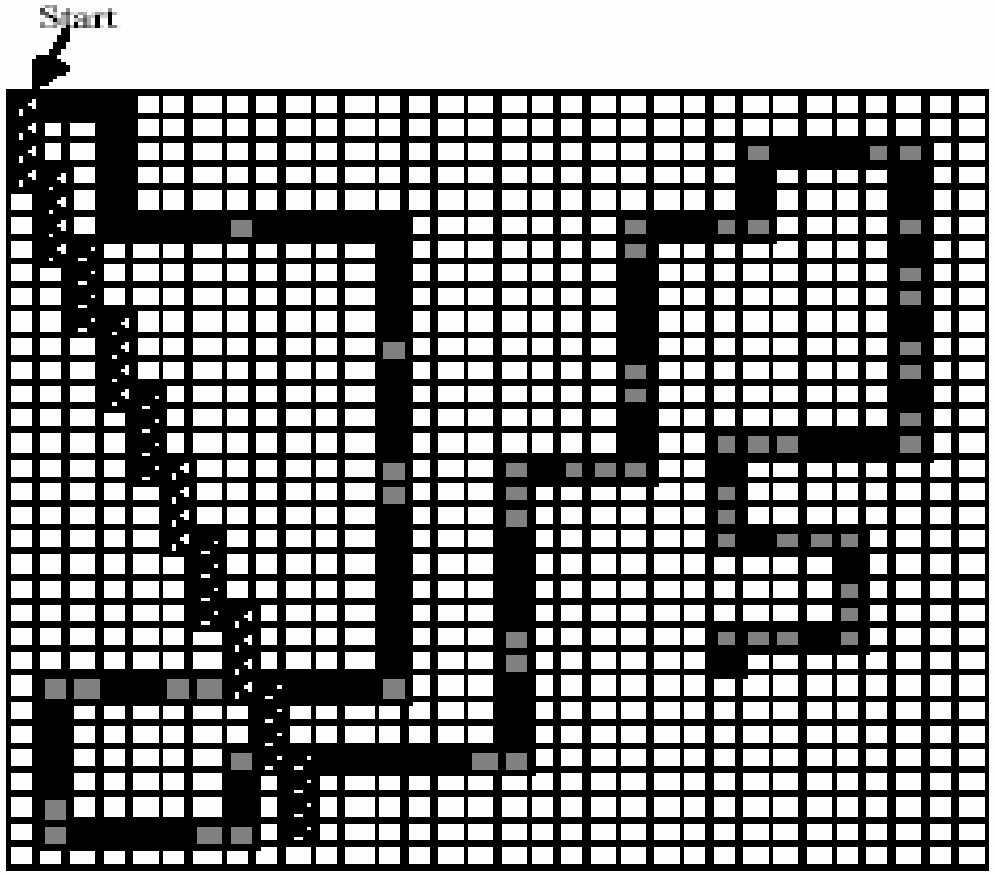
```
(SIC (DERECHA) (IZQUIERDA))
....
(SIC (DERECHA) (DERECHA))
```

- Contiene condicional pero usa la información mal

```
(SIC (PROGN2 (IZQUIERDA) (IZQUIERDA) (AVANZA))… )
```

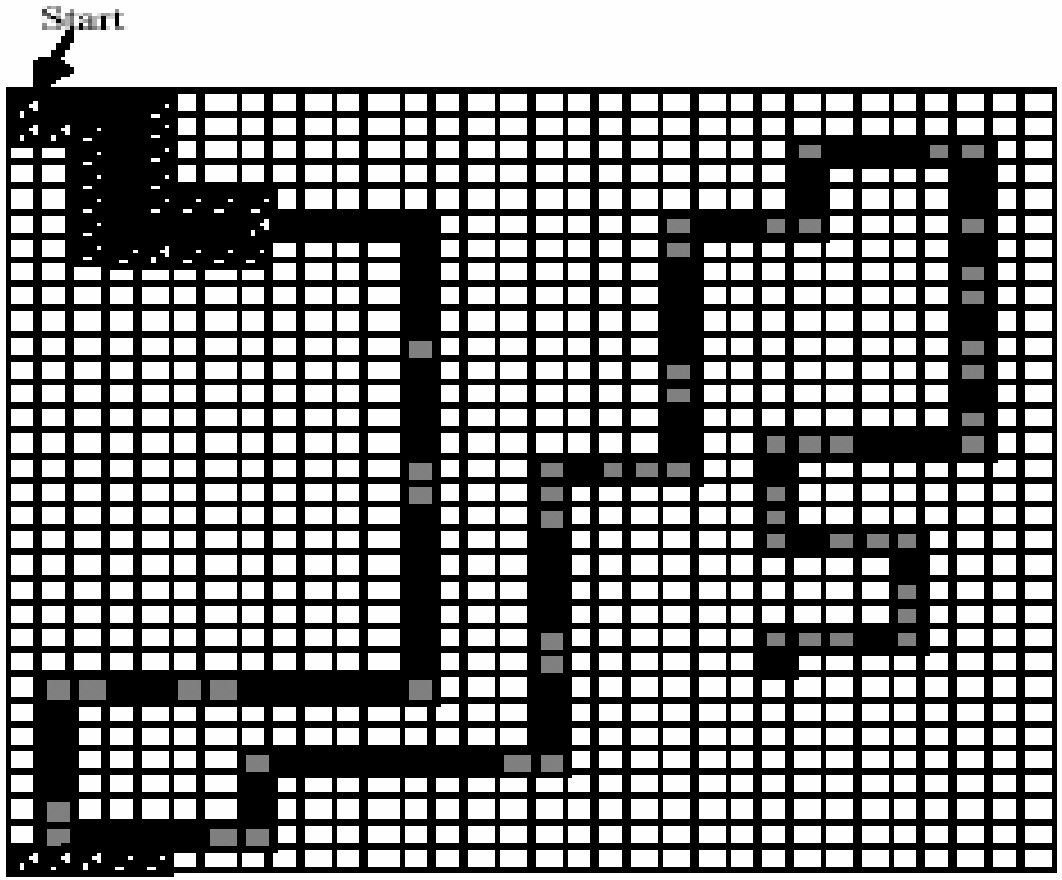
Ejemplo de la Hormiga

- (PROGN3 (DERECHA) (PROGN3 (AVANZA) (AVANZA) (AVANZA)) (PROGN2 (IZQUIERDA) (AVANZA)))
  - Paso 1 :D,A,A,A
  - Paso 2 :I,A
  - Paso 3 : Goto Paso1

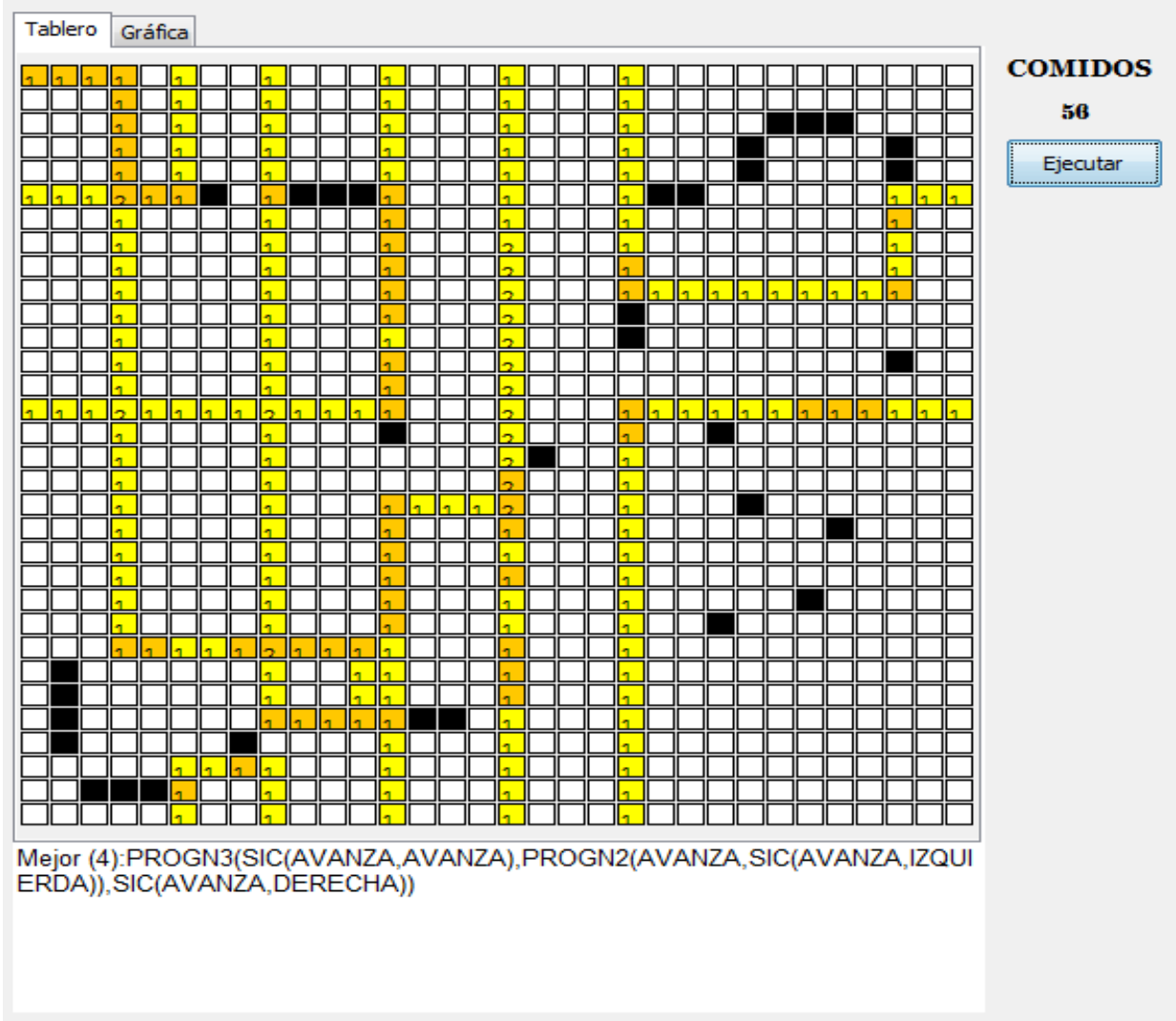


Estrategia esquivadora

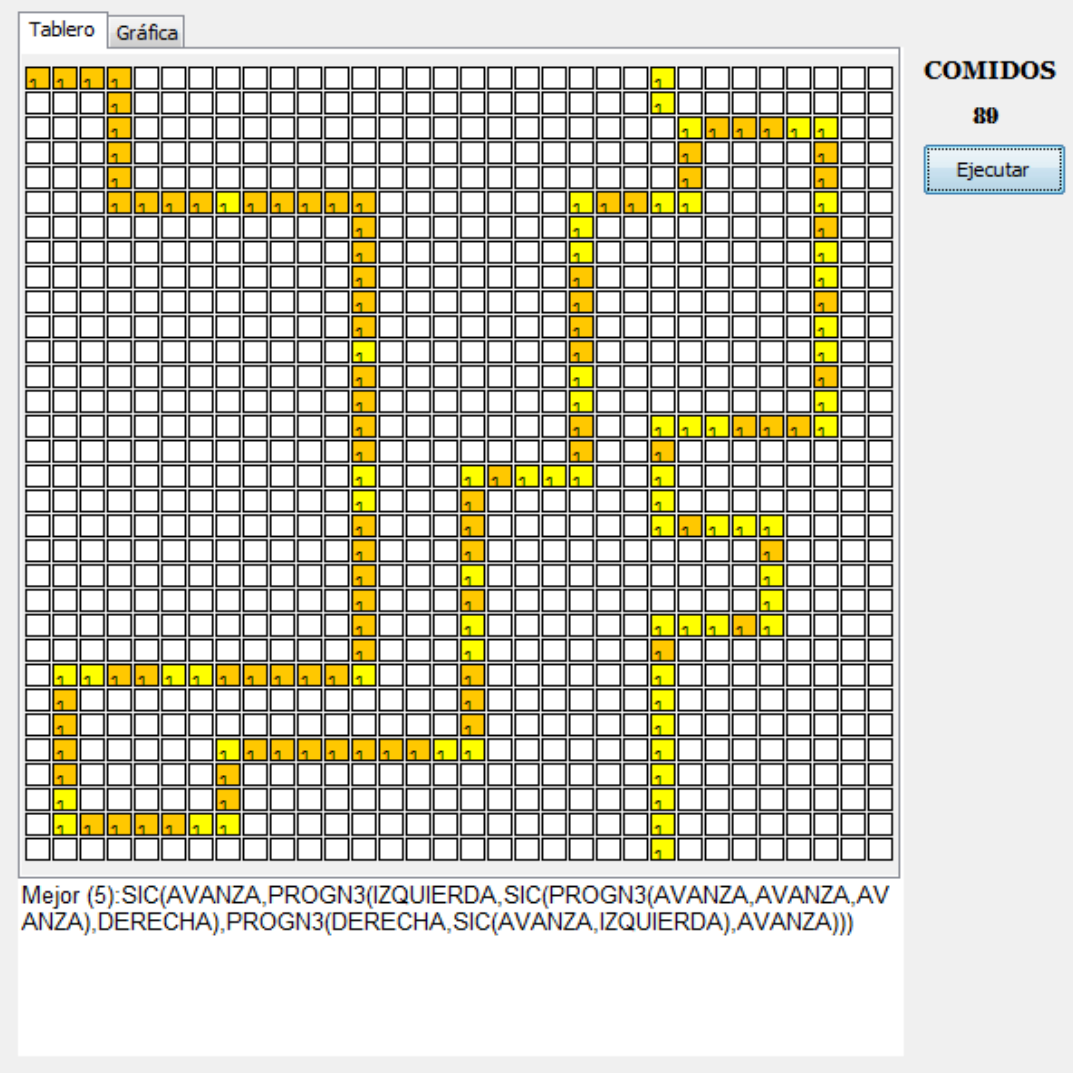
- (SIC (DERECHA) (SIC (DERECHA) (PROGN2 (AVANZA) (IZQUIERDA))))

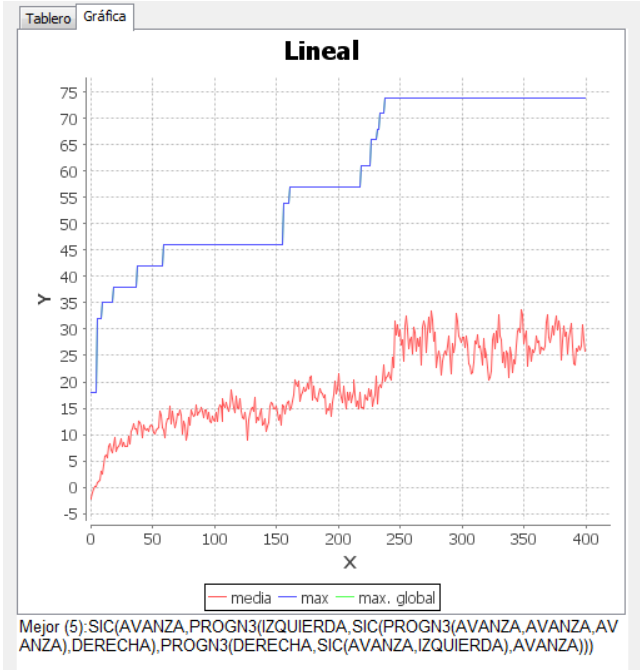


Ejemplos


Tablero
Gráfica
COMIDOS
56
Ejecutar
Mejor (4):PROGN3(SIC(AVANZA,AVANZA),PROGN2(AVANZA,SIC(AVANZA,IZQUIERDA)),SIC(AVANZA,DERECHA))

Ejemplos


Tablero
Gráfica
COMIDOS
89
Ejecutar
Mejor (5):SIC(AVANZA,PROGN3(IZQUIERDA,SIC(PROGN3(AVANZA,AVANZA,AVANZA),DERECHA),PROGN3(DERECHA,SIC(AVANZA,IZQUIERDA),AVANZA)))


Tablero
Gráfica
Lineal
Y
X
media
max
max. global
Mejor (5):SIC(AVANZA,PROGN3(IZQUIERDA,SIC(PROGN3(AVANZA,AVANZA,AVANZA),DERECHA),PROGN3(DERECHA,SIC(AVANZA,IZQUIERDA),AVANZA)))