

Programación genética

Algoritmos bioinspirados
y computación evolutiva

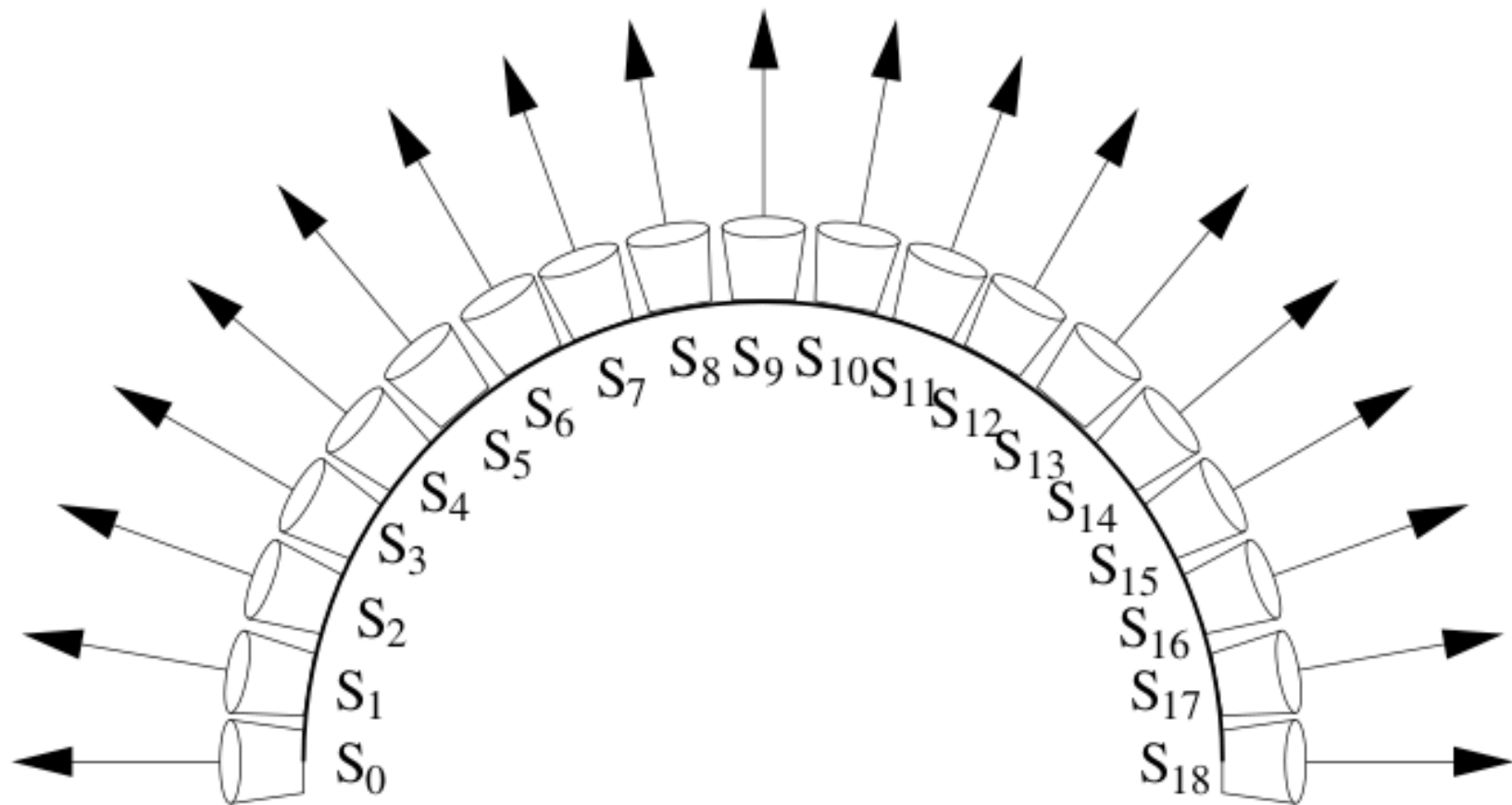
Ambientes dinámicos

- Algoritmos genéticos
 - Genomas muy largos
 - Diploidía / poliploidía
- Programas genéticos

TORCS



TORCS



TORCS: sensores

α	$[-\pi, \pi]/\text{rad}$	orientation of car relative to the current street orientation
distFromStart	$[0, \infty]/\text{m}$	distance from start line to current position of car measured along
distRaced	$[0, \infty]/\text{m}$	total distance traveled since beginning of race
t_{curLap}	$[0, \infty]/\text{s}$	elapsed time on current lap
t_{lastLap}	$[0, \infty]$	time elapsed for last lap
racePos	1, 2, ...	current rank of car in race
damage	$[0, \infty]/\text{point}$	total damage incurred to car
fuel	$[0, \infty]/\text{l}$	fuel left in fuel tank
gear	$\{-1, 0, 1, \dots, 6\}$	current gear position (-1: backwards, 0: neutral)
rpm	$[2000-7000]/\text{rpm}$	rounds per minute of the motor
d_y	$[-1, 1]$	displacement of car from center of track (normalized to 1)
v_x	$[-\infty, \infty]/\frac{\text{km}}{\text{m}}$	velocity of car in track direction
v_y	$[-\infty, \infty]/\frac{\text{km}}{\text{m}}$	velocity of car perpendicular to track direction
$v_{\text{wheel},i}$	$[0, \infty]/\frac{\text{rad}}{\text{s}}$	velocity of wheel $i \in \{1, 2, 3, 4\}$
S_i	$[0, 100]/\text{m}$	dist. to track boundary measured by 19 sensors $i \in \{0, \dots, 18\}$ as
O_i	$[0, 100]/\text{m}$	distance to opponents measured by 18 sensors $i \in \{1, \dots, 18\}$

TORCS: controles

gas pedal	$[0, 1]$
brake pedal	$[0, 1]$
steering	$[-1, 1]$
gear	-1, 0, 1,..., 6
meta control	0, 1

acceleration
brake (0: don't brake, 1: full brake)
orientation of steering wheel (-1: maximum left, 1: maximum right)
shift into gear as specified
meta control flag (0: do nothing, 1: restart race)

Programas como árboles

- $(2+x+y) * (1-\text{sqrt}(18/(x+y))) / ((x-y)^2 + 30) * (x^2 - y^2)$

$$\frac{(2+x+y) \cdot \left(1 - \sqrt{\frac{18}{x+y}}\right)}{((x-y)^2 + 30) \cdot (x^2 - y^2)}$$

Programas como árboles

- $$\frac{(2+x+y) * (1-\text{sqrt}(18/(x+y)))}{(((x-y)^2 + 30) * (x^2-y^2))}$$
- $$\frac{(+ 2 x y) (- 1 (\text{sqrt} (/ 18 (+ x y))))}{(+ (\text{expt} (- x y) 2) 30) (- (\text{expt} x 2) (\text{expt} y 2)))}$$

Programas como árboles en R

```
ee <- expression (  
  (2+x+y) * (1-sqrt(18/(x+y))) /  
  (((x-y)^2 + 30) * (x^2-y^2)) )
```

```
as.list (ee[[1]])
```

```
[[1]] → `/`
```

```
[[2]] →
```

```
(2 + x + y) * (1 - sqrt(18/(x + y)))
```

```
[[3]] →
```

```
((x - y)^2 + 30) * (x^2 - y^2)
```

Programas como árboles

- `if (a>b) {x<-a;cat(b)} else stop()`
- `(if (> a b)
 (progn (setq x a)
 (princ b))
 (error))`

Programas como árboles en R

```
ee <- expression (  
  if (a>b) {x<-a;cat(b)} else stop() )
```

```
as.list (ee[[1]][[3]])
```

```
[[1]] → `{`
```

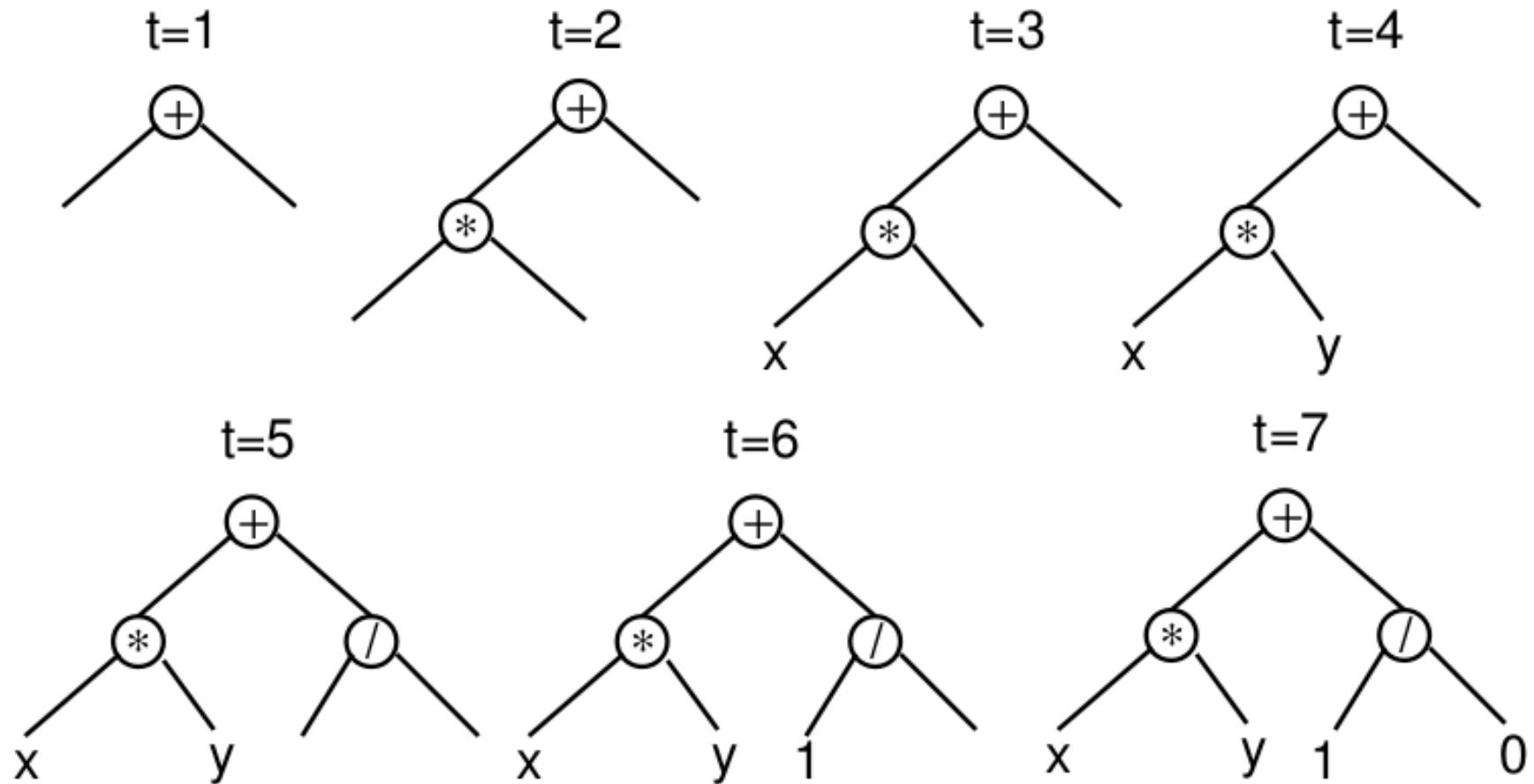
```
[[2]] → x <- a
```

```
[[3]] → cat(b)
```

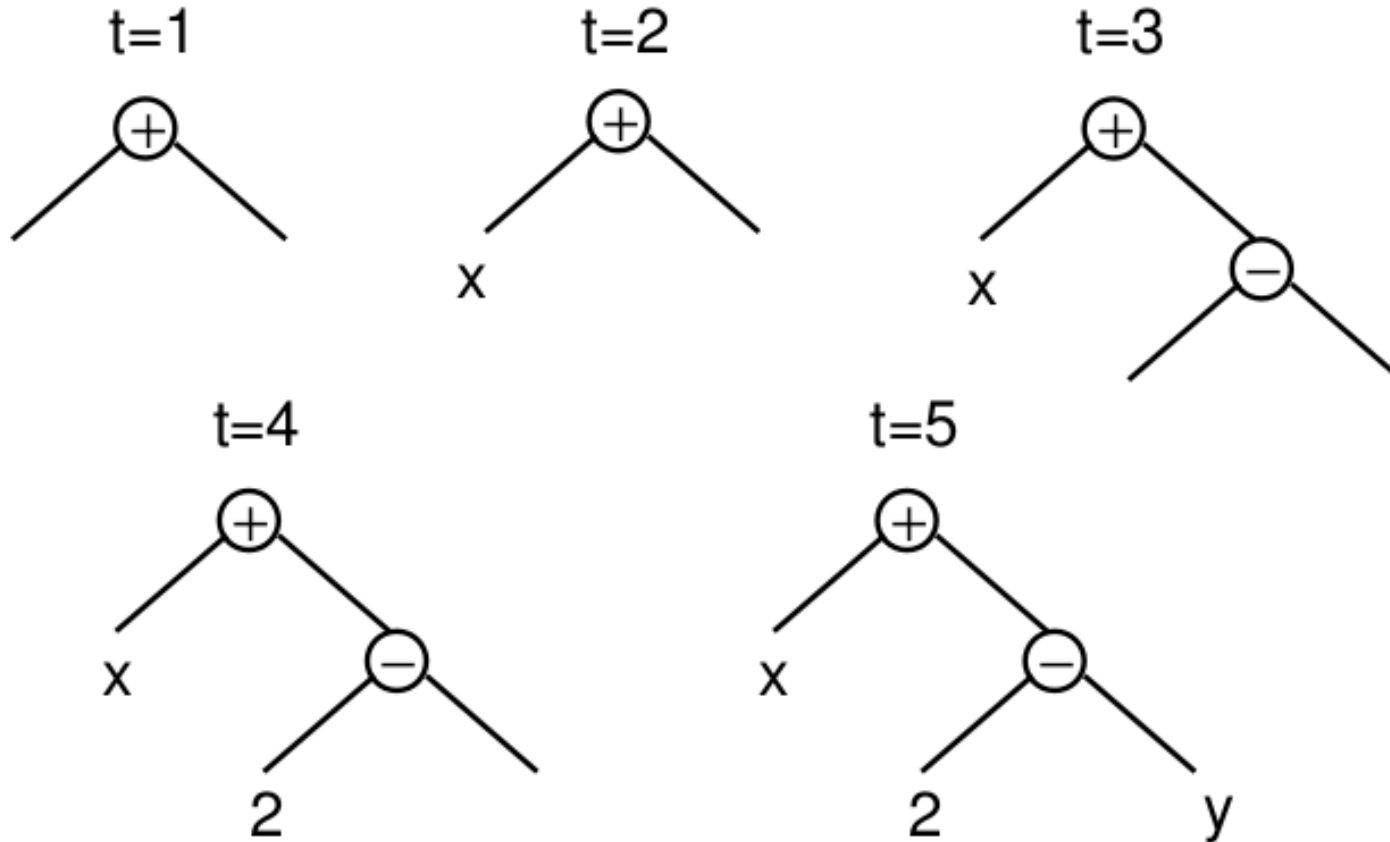
Elementos de un árbol

- Operadores
 - $+$, $-$, $*$, $/$...
 - Ifelse; $==$, not, $>$...
- Terminales
 - Constantes: 0, 1, π ...
 - Aleatorios: runif...
 - Variables: x, y...

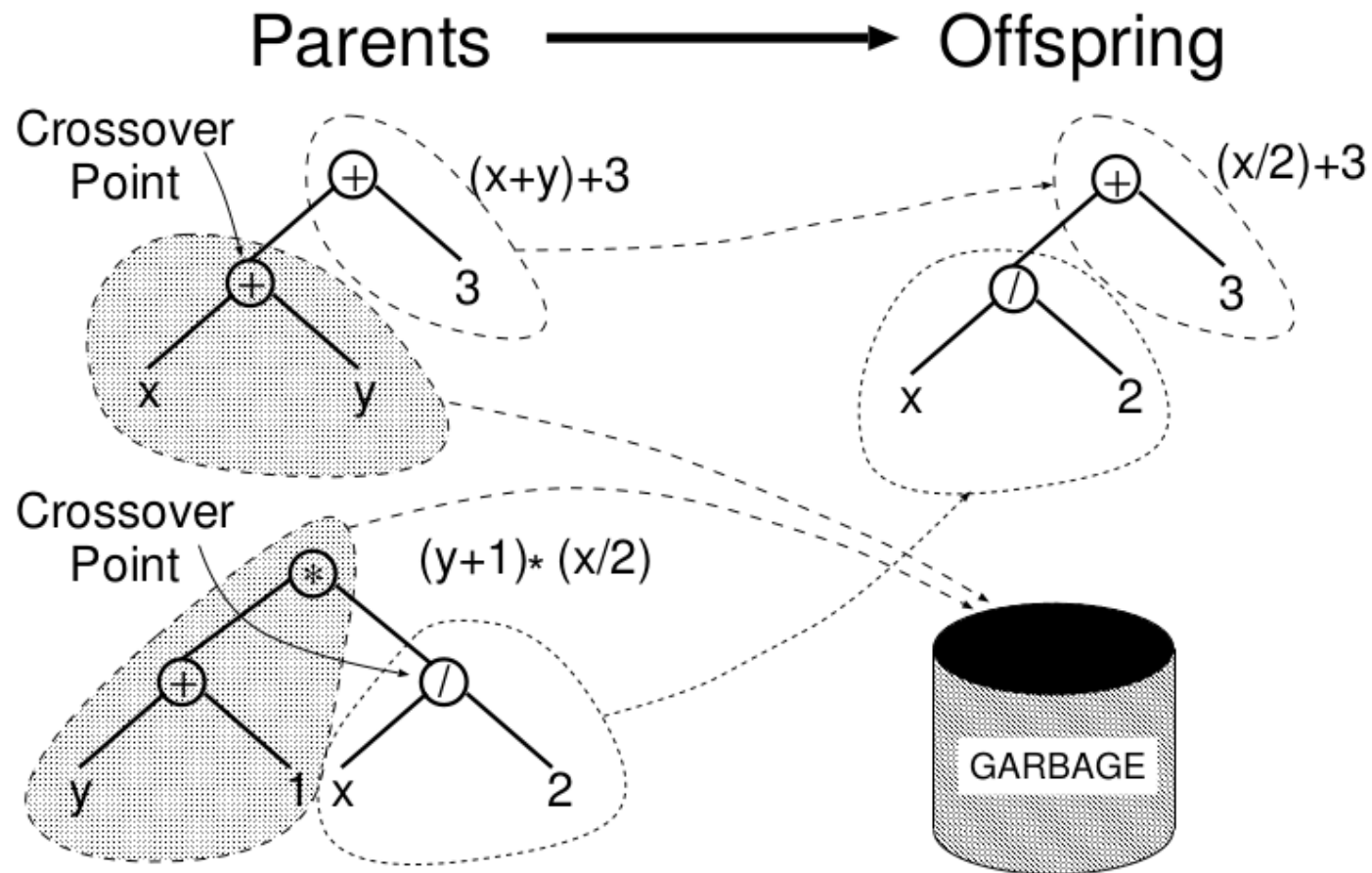
Inicializar población (*full*)



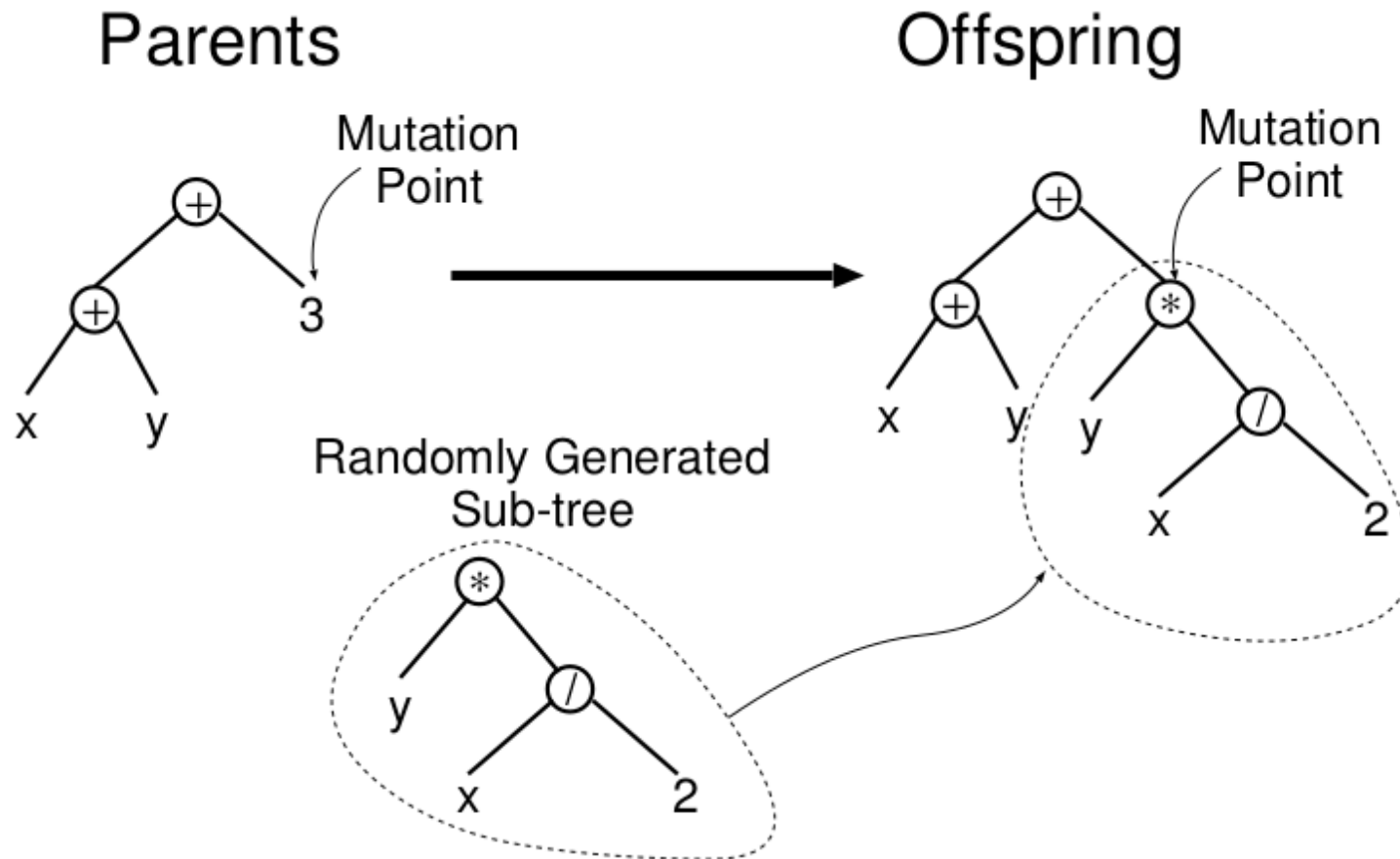
Inicializar población (*grow*)



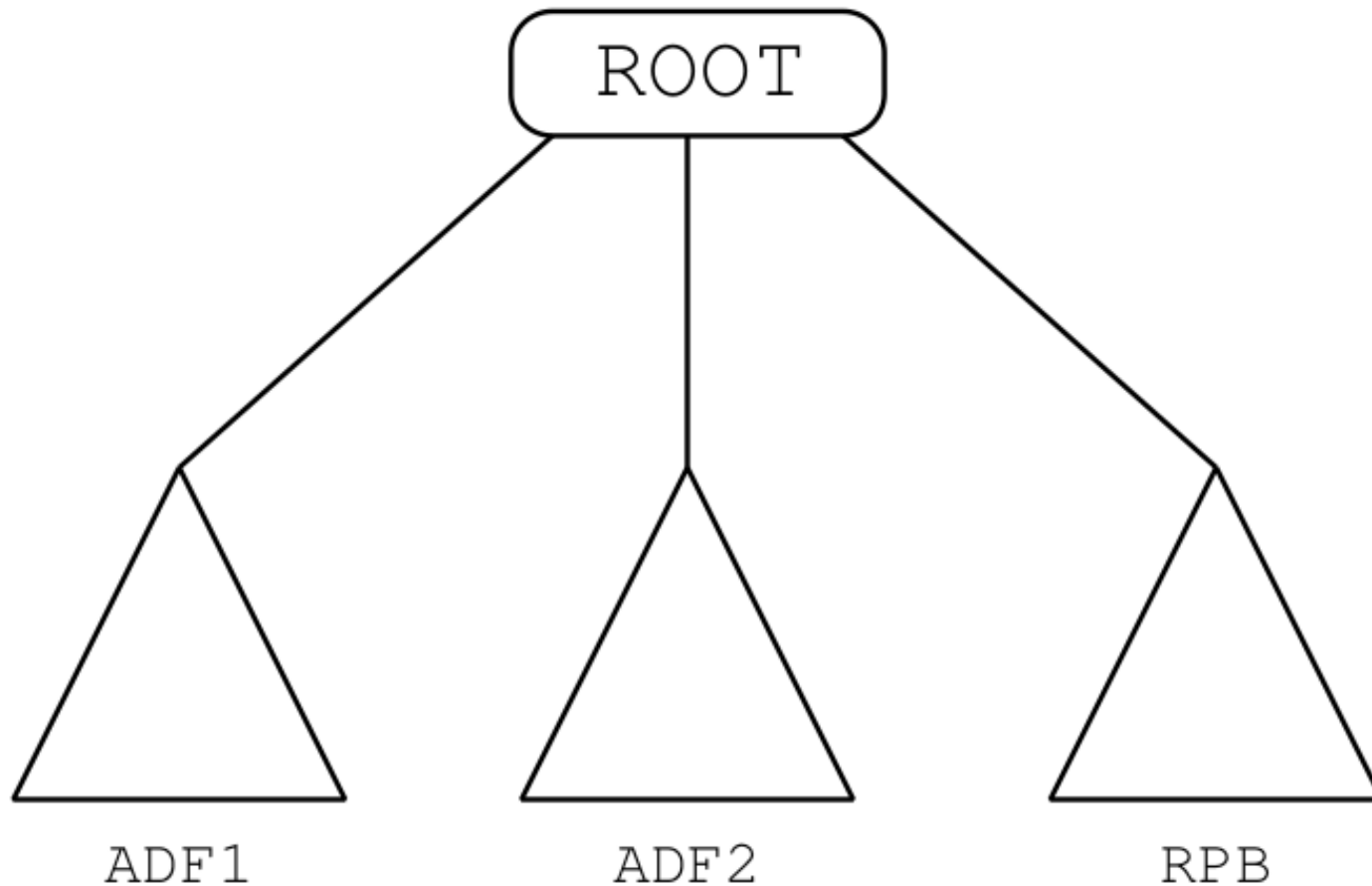
Recombinación (sobrecruce)



Mutación



P.G. avanzada: A.D.F.



P.G. av.: restricciones estructurales

- Estructuras particulares
 - p.ej. Periódicas: $a \cdot \sin(b \cdot t)$
- Tipos estrictos en argumentos
 - p.ej. `ifelse(<logical>, <numeric>, <numeric>)`
 - Disponibles en RGP

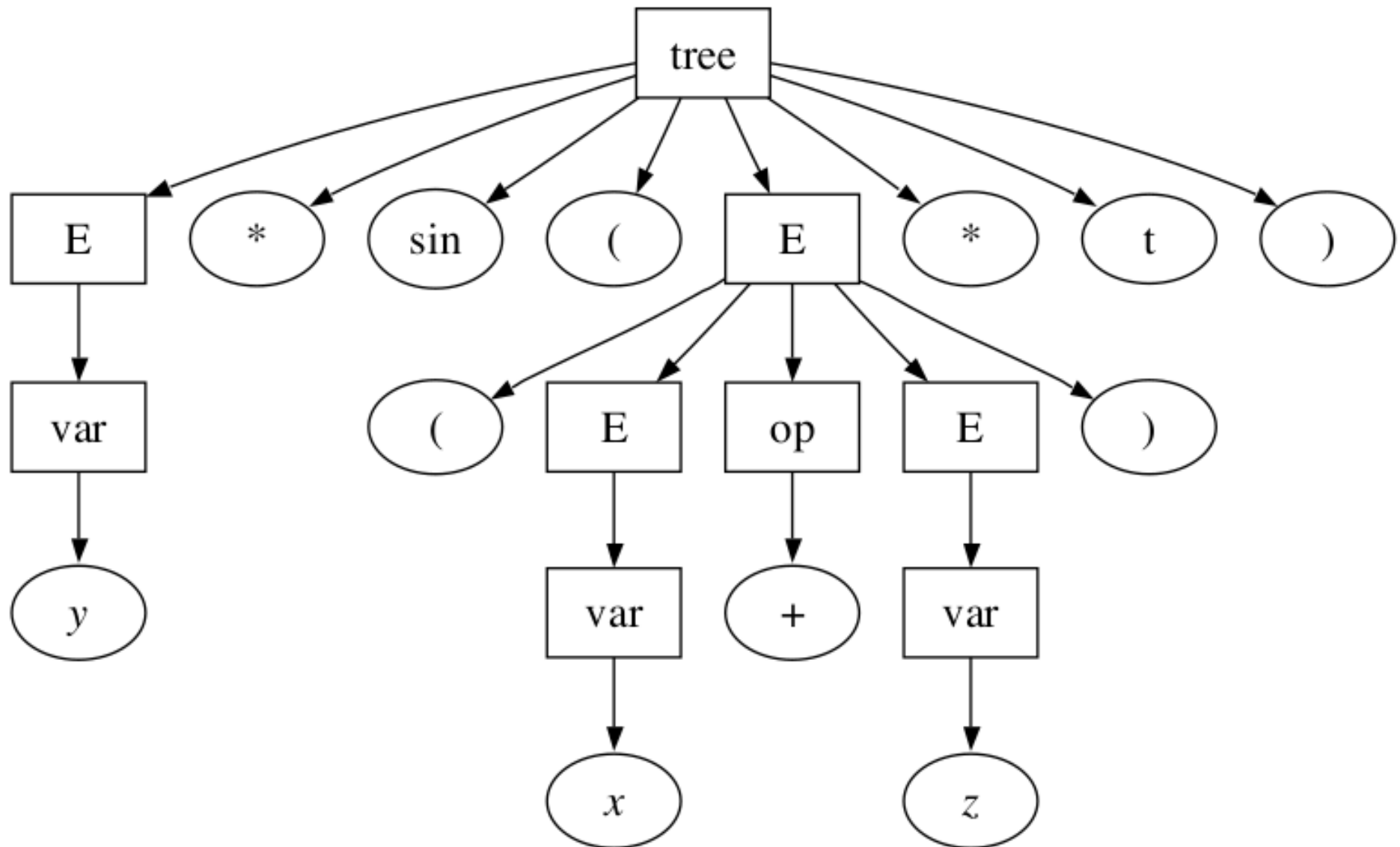
P.G. av.: restricciones gramaticales

- Gramática

- Árbol $::= E . \text{sen} (E . t)$
- $E ::= \text{var} \mid (E \text{ op } E)$
- $\text{op} ::= + \mid - \mid . \mid :$
- $\text{var} ::= x \mid y \mid z$

- Recombinación sólo posible entre nodos con el mismo símbolo gramatical (no terminal)

P.G. av.: restricciones gramaticales



P.G. avanzada: PushGP

- Programación autoconstructiva con tipado estricto
- Tipos: booleanos, enteros, coma-deslizante... y árbol de código
- Cada tipo: pila propia
- Soporta recursión y módulos (ADF)
- Un programa (o parte de sí mismo) puede meterse en la pila al evolucionar
- Operadores de recombinación y mutación evolucionan en esa pila

TORCS: árbol del volante

Name	Arguments	Description
ERC1	0	ephemeral random constant with range $[-1, 1]$
ERC150	0	ephemeral random constant with range $[-150, 150]$
c_p	0	constant for hand-crafted proportional controller $c_p = -0.0234$
LR0	0	average difference between left and right track sensors $\frac{1}{2}(S_{15} + S_{16})$
abs	1	identity function (should have been absolute value)
+	2	sum of both arguments
-	2	difference of both arguments
*	2	product of both arguments
/	2	protected division

TORCS: árbol de acelerador/freno

Name	Arguments	Description
ERC1	0	ephemeral random constant with range $[-1, 1]$
ERC50	0	ephemeral random constant with range $[-150, 150]$
c_1	0	first constant (used by hand-crafted gas/brake controller)
c_2	0	second constant (used by hand-crafted gas/brake controller)
LR1	0	difference between left and right front facing track sensors
S_9	0	front facing sensor
v_x	0	velocity of car
abs	1	identity function (should have been absolute value)
+	2	sum of both arguments
-	2	difference of both arguments
*	2	product of both arguments
/	2	protected division

Ejemplín en R

- Objetivo: hallar fórmula para calcular el área de un círculo a partir del radio

- Operadores:

```
.operators. <- c("+", "-", "*", "/")
```

Aridad binaria para todos ellos

- ```
random.el <- function (lista)
 sample (lista, 1) [[1]]
```



```
random.form <- function (operators) {
 lista <- list (random.elc (operators))
 for (i in 1:2) # aridad
 lista <- c (lista,
 {
 random.nr <- runif (1)
 if (random.nr < .5)
 list (random.form (operators))
 else if (random.nr < .75)
 list (runif (1, 0, 10))
 else list ("=input=")
 })
 lista }
```

# Generar árbol aleatorio

- Profundidad esperada: 2
- Profundidad: no limitada; sí en **esta versión**:  
`random.form ( . . . , max.depth = 4 )`
- Aleatorios entre 0 y 10 (buscamos  $\pi$ )
- `=input=` es una variable (`x`, el radio del círculo)
- Inicialización “grow”  
`árbol <- random.form (.operators.)`

# Generar árbol aleatorio

- Visualizar el árbol:

```
escribe.lista (árbol)
```

```
traverse.nodes.example (árbol)
```

```
texto <- traverse.nodes.infiijo (árbol)
```

```
cat (texto, ";", file="/tmp/texto.mac")
```

```
system ("maxima -b /tmp/texto.mac")
```

- Evaluarlo:

```
x <- 10 ; eval (parse (text = texto))
```

```
run.form (árbol, 10)
```

```
run.form.insegura <- function (form, input) {
 if (is.atomic (form))
 if (form == "=input=")
 input
 else form
 else do.call (form[[1]],
 lapply (form[-1],
 run.form.insegura,
 input)) }

run.form <- function (form, input) {
 intento <- try (run.form.insegura (form, input))
 if (inherits (intento, "try-error")) NA
 else intento }
}
```

# Población inicial

```
for (i in
 create.initial.population
 (.operators., 10))
{
 escribe.lista(i)
 cat("\n")
}
```

```
create.initial.population <-
 function(operators, size=100)
 lapply (1:size,
 function (i) random.form (operators))

fitness <-
 function (form, fitness.fn, test.input)
 prod (sapply (test.input,
 function (input) {
 output <- run.form (form, input)
 target <- fitness.fn (input)
 difference <- abs (target - output)
 fitness.value <- 1 / (1 + difference)
 })))
```

# Adaptación

```
árbol <- random.form (.operators.)
escribe.lista (árbol)
fitness (árbol,
 function (r) pi * r^2,
 c (0, 1, 2))
puede ser NA por división por cero
```

# Recorriendo las ramas

- `n.nodes (árbol) # nº nodos (incluida raíz)`
- `random.node (árbol) # evita la raíz`
- `replace.node (árbol, 2, 'XXX')`



# Recombinación y mutación

- `árbol1 <- random.form (.operators.)`
- `árbol2 <- random.form (.operators.)`
- `escribe.lista (`  
    `cross.over (árbol1, árbol2, TRUE))`
- `escribe.lista (`  
    `mutate (árbol, .operators., TRUE))`

# Evaluar la población

```
población <-
 create.initial.population
 (.operators., 10)
evaluada <- evaluate.population
 (población,
 function (r) pi*r^2,
 c (0, 1, 2))
evaluada[[1]]$fitness # la mejor
```

```

advance.generation <- function (pop, fitness.fn,
 operators, test.input, max.population=100) {
 epop <- evaluate.population(pop,fitness.fn,test.input)
 cat ("Best fitness of current population:",
 epop[[1]]$fitness, "\n")
 res <- list()
 for (plist in head (epop, max.population)) {
 fitness <- plist$fitness
 form <- plist$form
 res <- c (res, list (form))
 if (runif (1) < fitness)
 res <- c (res, list (if (runif (1) < .9)
 cross.over (form,
 random.elit(epop)$form)
 else mutate (form, operators)))
 if (runif (1) < 0.02)
 res <- c (res, list (random.form (operators))) }
 res }

```

```

population <- create.initial.population (.operators.,
 100)

for (i in 1:500) {
 cat("[",i,"] ")
 population <- advance.generation (population,
 function(r) pi*r*r,
 .operators.,
 list(0,1,-2))}

best.form <- function ()
 evaluate.population (population,
 function (r) pi * r^2,
 list (0, 1, -2)) [[1]]

traverse.nodes.example(best.form())$form)
traverse.nodes.infijo(best.form())$form) # para Maxima

```

# Bibliografía

- A field guide to genetic programming
- On the programming of computers by means of natural selection