Fundamentos de la Evolución Gramatical

José Manuel Colmenar

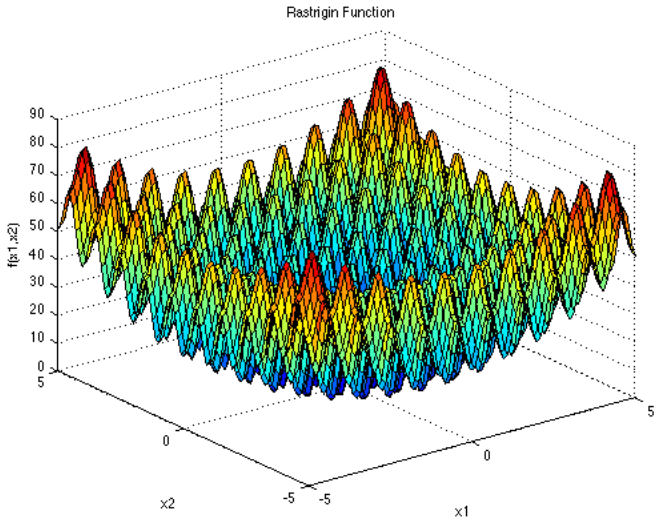
Adaptive and Bioinspired Systems Research Group

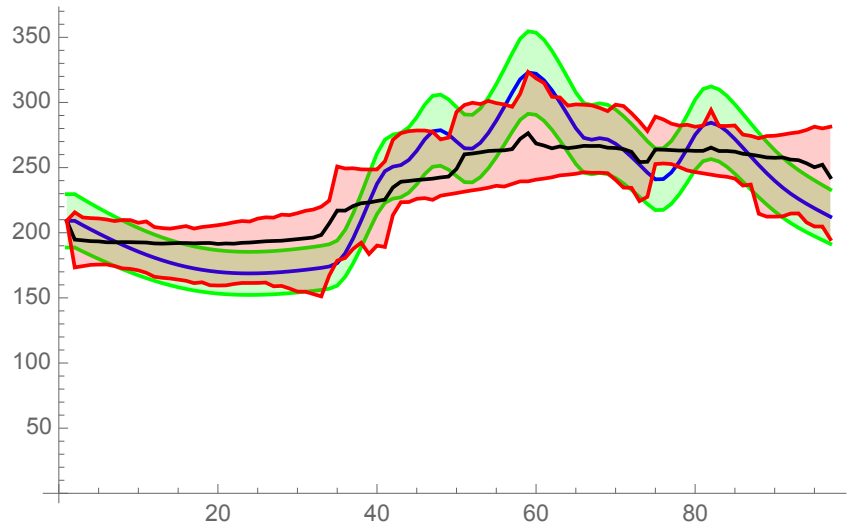
Universidad
Rey Juan Carlos

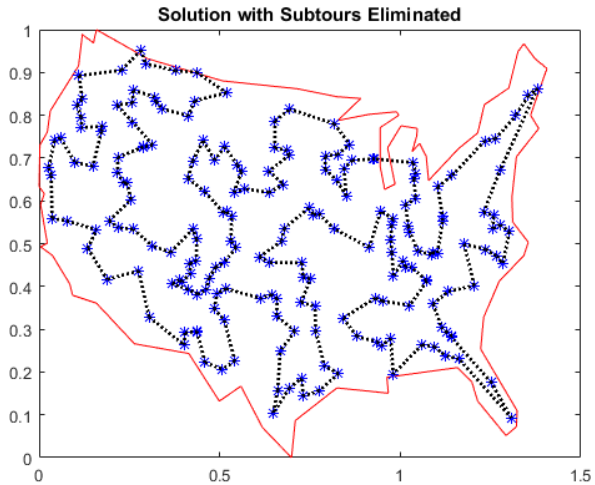
Índice

1. Introducción
2. Algoritmos Evolutivos
   - Características
   - Algoritmos Genéticos
   - Programación Genética
3. Evolución Gramatical
   - Introducción
   - Gramáticas
   - Decodificación
   - Operadores
4. Conclusiones
   - Resumen
   - Aplicaciones Prácticas

Problemas de Optimización

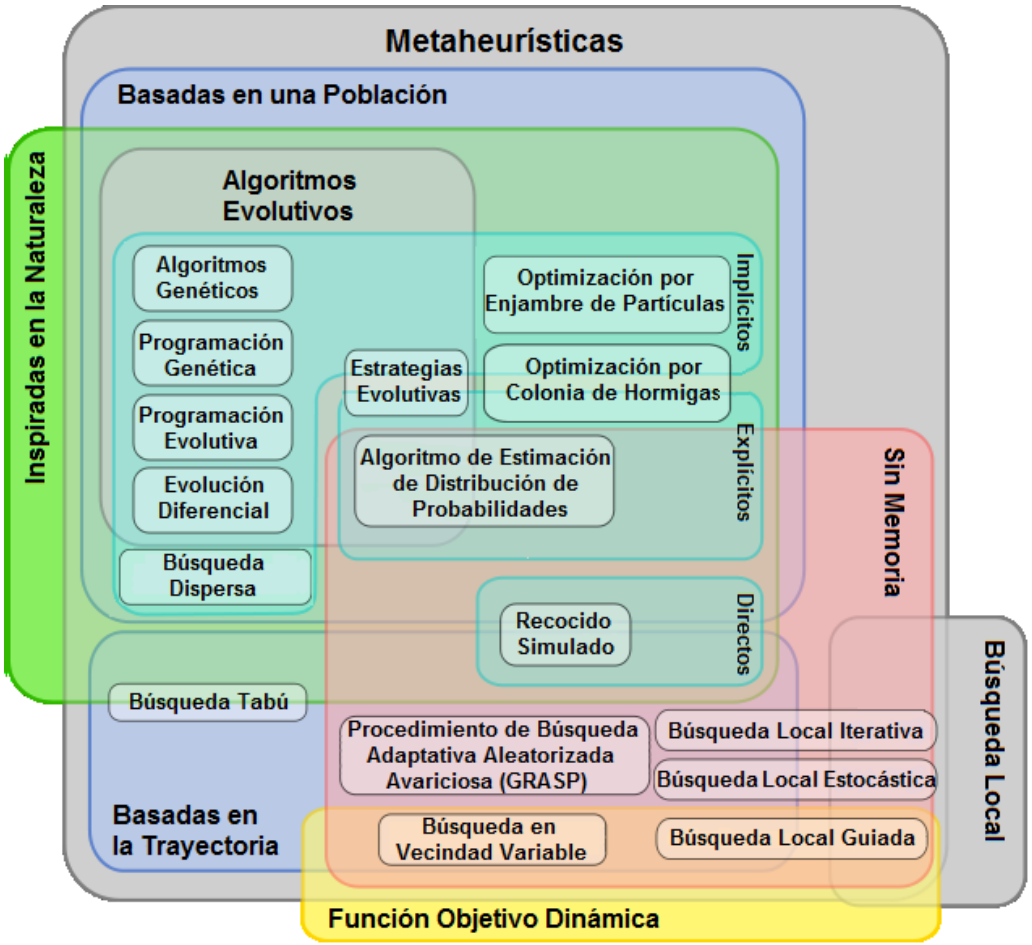






- Problemas "difíciles": NP-completo, NP-duro, ...
- Gran espacio de soluciones.
- Lenta evaluación.
- Fuerza bruta no es efectiva.

Metaheurísticas



"Metaheurísticas clasificación" de Johann "nojhan" Dréo and myself (translation to Spanish)
Disponible bajo la licencia CC BY-SA 3.0 vía Wikimedia Commons
https://commons.wikimedia.org/wiki/File:Metaheur %C3 %ADsticas_clasificaci %C3 %B3n.png

Algoritmos Evolutivos

Esquema general

Inicializar **población** aleatoriamente (o no).
Evaluar población.
Repetir hasta cumplir la **condición de terminación** {

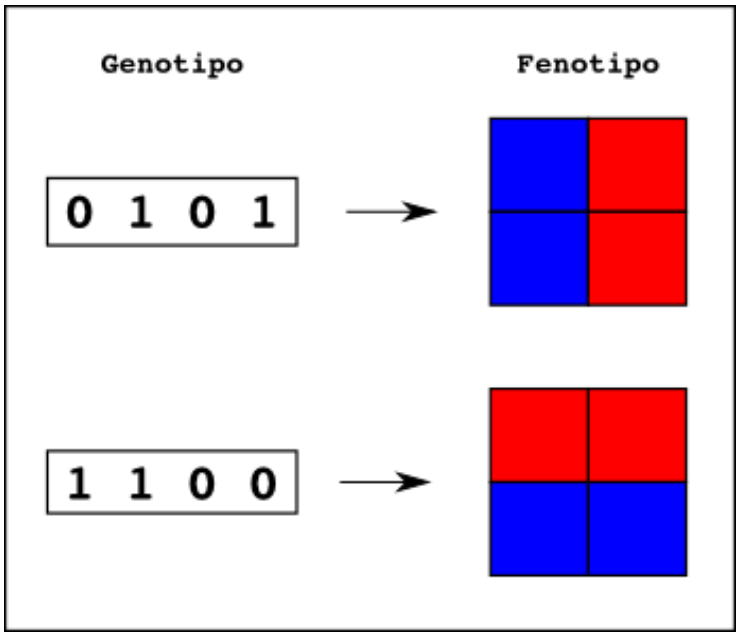
1. **Selección** de progenitores.
2. **Recombinación** de progenitores.
3. **Mutación** de descendientes.
4. **Evaluación** de nuevos individuos.
5. Determinación de la **nueva población**.

}
Selección del **mejor individuo** de la población final.

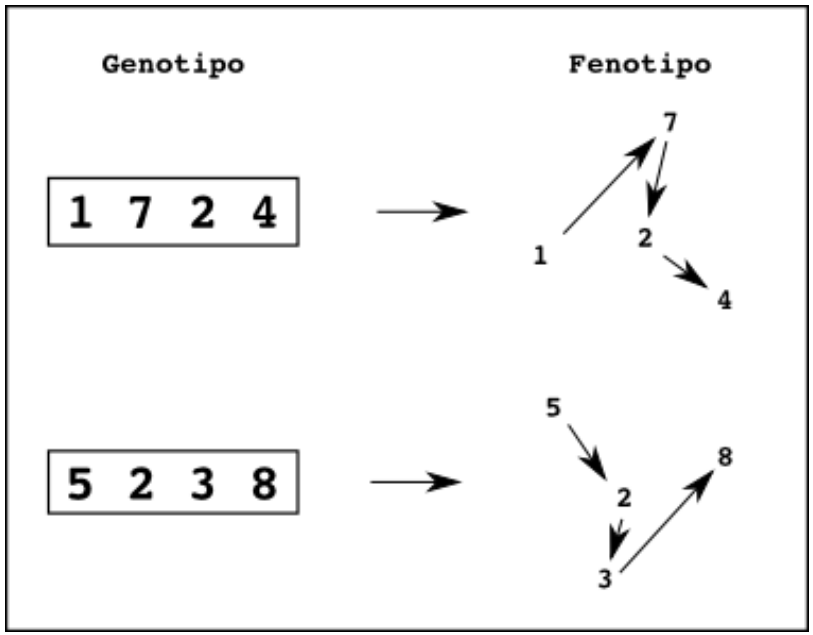
- *Survival of the fittest*.
- La **diversidad** ayuda a mejorar la exploración.
- Con el paso de las generaciones la población **converge**.
- La población puede empeorar si no se mantiene el **elitismo**.

Algoritmos Genéticos

- Se basan en la codificación del ADN en los **cromosomas**.
- El **genotipo** es la información codificada de un individuo.
- El **fenotipo** es la expresión del genotipo.



Codificación binaria

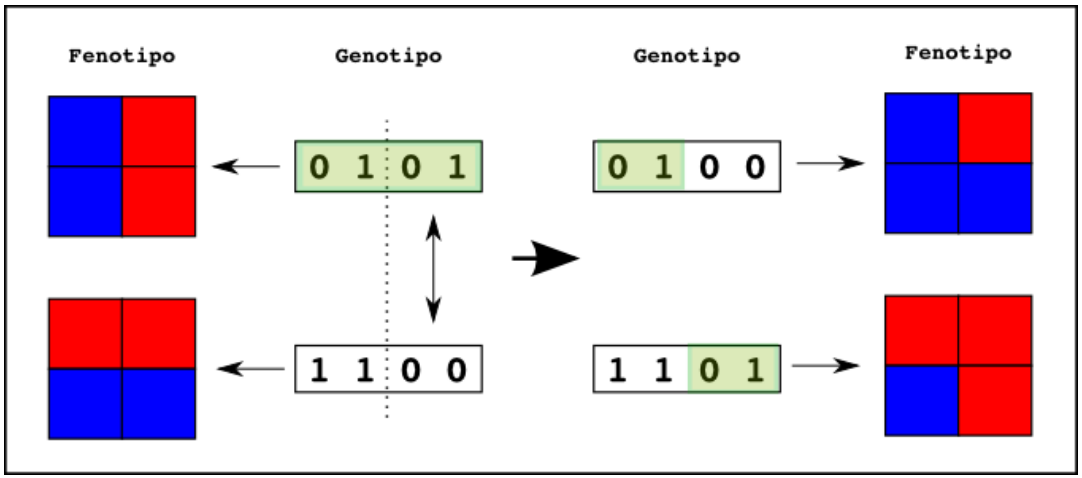


Codificación entera

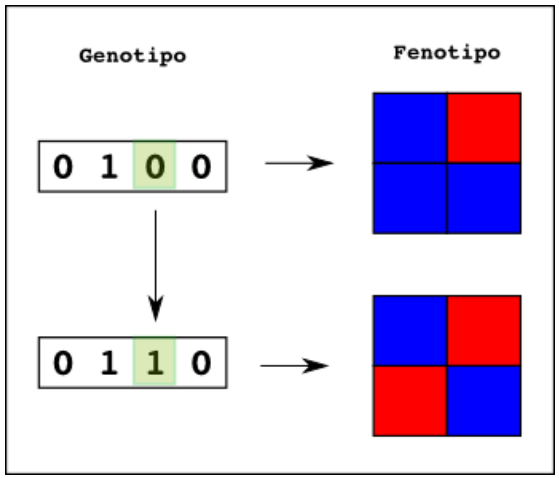
Operadores Genéticos

- Los operadores genéticos **se aplican sobre el genotipo**.
- El **efecto** se produce **en el fenotipo**.
- Se aplican a los individuos en base a una **probabilidad**.
- Existen diversos tipos de cruce cuya aplicación depende del problema o la codificación:
  - Cruce de 1 punto, N puntos, cíclico, uniforme ...
  - Combinaciones de varios progenitores.

Cruce

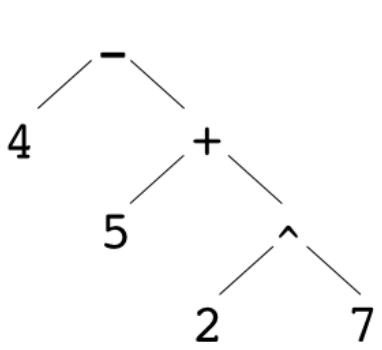


Mutación

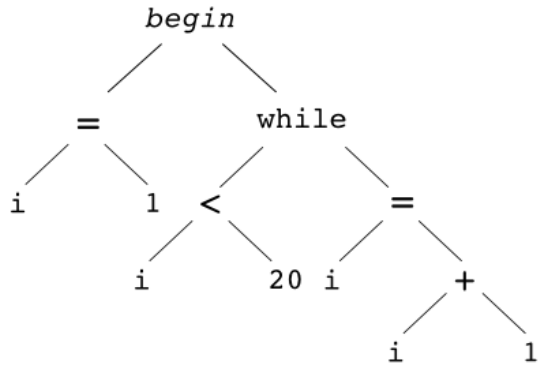


Programación Genética

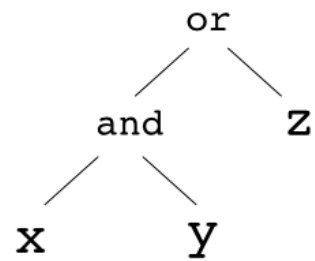
- Los individuos son árboles:
  - Expresiones artiméticas
  - Programas
  - Fórmulas lógicas
  - ...




```
4 - 5 + 2^7
```




```
i = 1
while (i < 20) {
   i = i +1;
}
```

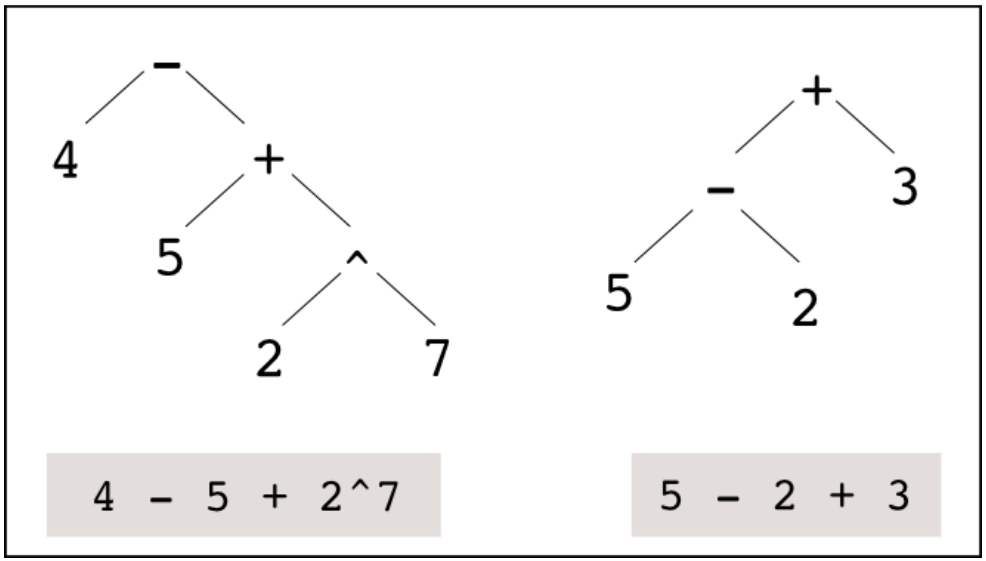



```
x and y or z
```

Programación Genética

- La **recombinación** consiste en **intercambiar sub-árboles**.
- La **mutación** consiste en **cambiar el árbol aleatoriamente**.

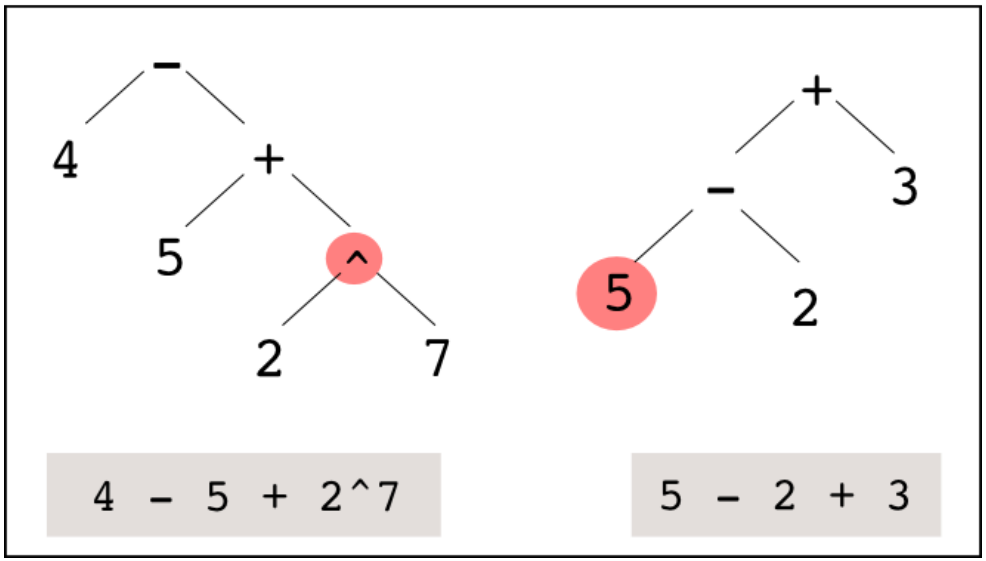
Recombinación



Programación Genética

- La **recombinación** consiste en **intercambiar sub-árboles**.
- La **mutación** consiste en **cambiar el árbol aleatoriamente**.

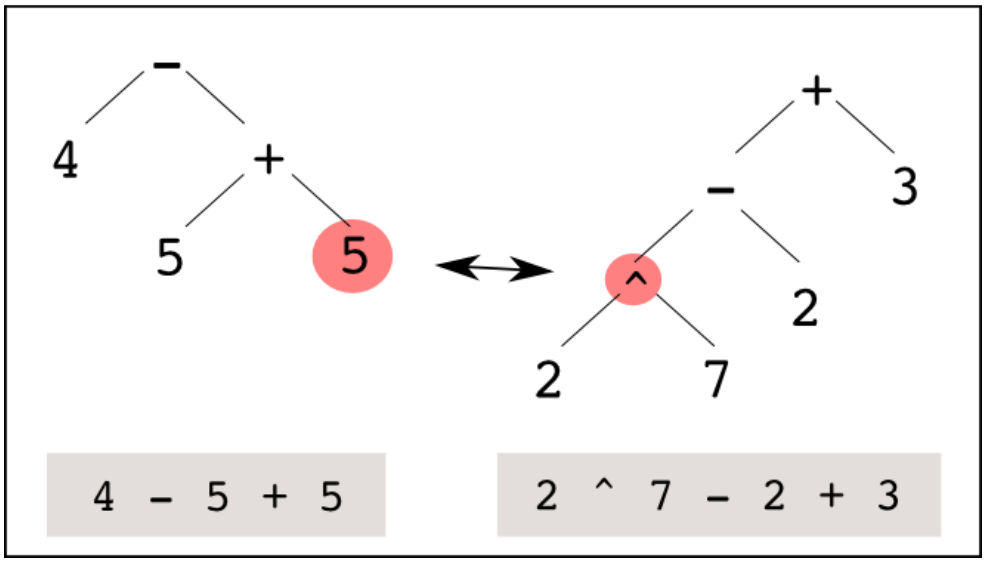
Recombinación



Programación Genética

- La **recombinación** consiste en **intercambiar sub-árboles**.
- La **mutación** consiste en **cambiar el árbol aleatoriamente**.

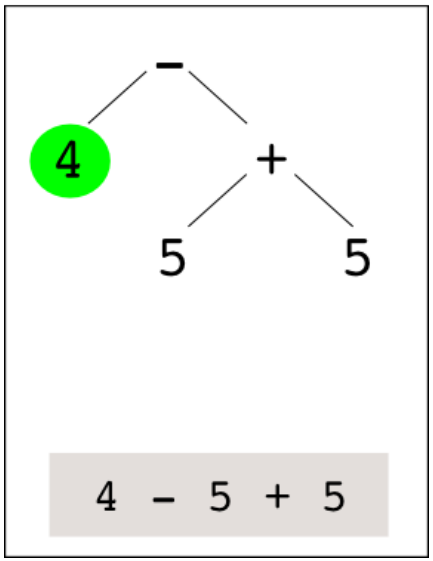
Recombinación



Programación Genética

- La **recombinación** consiste en **intercambiar sub-árboles**.
- La **mutación** consiste en **cambiar el árbol aleatoriamente**.

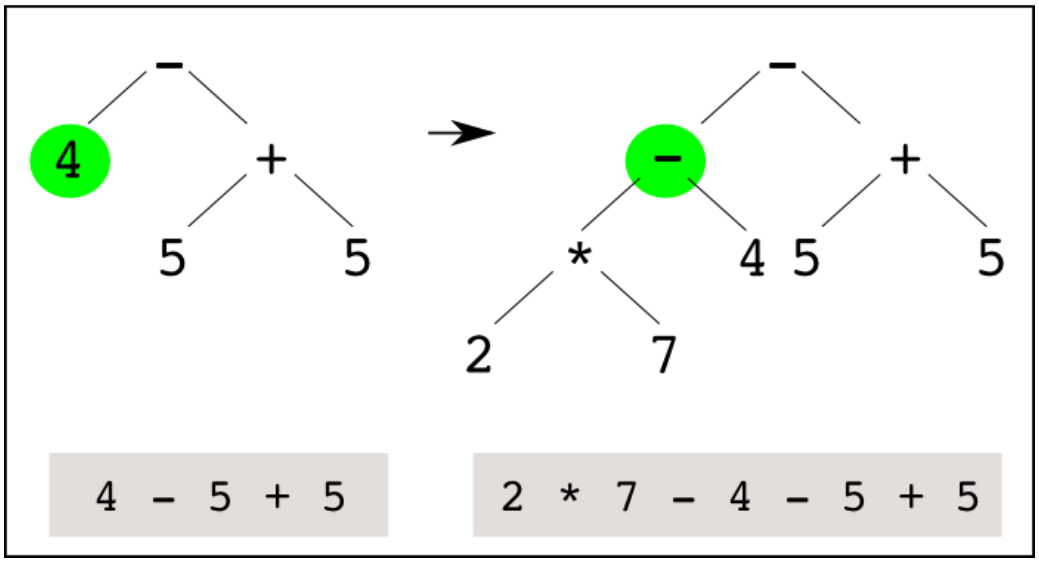
Mutación



Programación Genética

- La **recombinación** consiste en **intercambiar sub-árboles**.
- La **mutación** consiste en **cambiar el árbol aleatoriamente**.

Mutación



Programación Genética

- Su utilidad más práctica es la **obtención de modelos**.
- La codificación de árboles es compleja:
  - El tamaño de los cromosomas es variable.
  - Requiere análisis (*parsing*) o codificaciones alternativas.

- Introduce el problema del **bloating**:
  - El tamaño de los cromosomas crece a lo largo de las generaciones.
  - *Survival of the fattest*.
  - Se puede reducir penalizando el tamaño de los cromosomas.

Programación Genética

- Su utilidad más práctica es la **obtención de modelos**.
- La codificación de árboles es compleja:
  - El tamaño de los cromosomas es variable.
  - Requiere análisis (*parsing*) o codificaciones alternativas.

- Introduce el problema del **bloating**:
  - El tamaño de los cromosomas crece a lo largo de las generaciones.
  - *Survival of the fattest*.
  - Se puede reducir penalizando el tamaño de los cromosomas.

Características

Evolución Gramatical

Forma de programación genética basada en gramáticas.

Michael O'Neill and Conor Ryan. 2003. Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers, Norwell, MA, USA.

- Los individuos se codifican a través de **cromosomas con codificación entera** (como sucede en AGs).
- Permite aplicar **operadores de recombinación y mutación de AGs**.
- La **expresión de un genotipo** se realiza **decodificando por medio de la gramática**.
- Los fenotipos (programas) se evalúan como se hace en PG.

Gramática

Una gramática se define con los siguientes elementos:

- N: conjunto de elementos **no terminales**.
- T: conjunto de elementos **terminales**.
- S: símbolo inicial.
- P: conjunto de **reglas de producción**.

Beneficios de la gramática:

- Aporta **capacidad de expresión**.
- Permite **delimitar el espacio de búsqueda**.
- Puede **aportar conocimiento** fijando términos.

Gramática para máquina de estados

```
N = {<FSM>,<State>,<StateId>,<Output>,<Trans>}
T = {Reset,0,1,A,B,C,D}
S = <FSM>

I    <FSM>      ::= <State>
II   <State>    ::= <StateId><Output><Trans><Trans>
III  <Trans>    ::= <State> | Reset
IV   <Output>   ::= 0 | 1
V    <StateId>  ::= A | B | C | D
```

Gramática para expresiones matemáticas

```
N = {<func>,<expr>,<op>,<var>,<cte>,<dgt>,<preop>}
T = {(,),+,-,*,/,x,y,z,.,0..9,sin,cos,tan,exp}
S = <func>

I    <func>   ::= <expr> <op> <expr> <op> <expr>
II   <expr>   ::= <preop> (<expr>)
                 |(<cte> <op> <var>)
                 |<var>
III  <op>     ::= + | - | / | *
IV   <var>    ::= x|y|z|<cte>
V    <cte>    ::= <dgt><dgt>.<dgt><dgt>
VI   <dgt>    ::= 0|1|2|3|4|5|6|7|8|9
VII  <preop>  ::= sin | cos | tan | exp
```

Proceso de decodificación (Mapeo)

Cromosoma

15	200	22	17	123

<func>

I	<func>	::=	<expr> <op> <expr>
			| <preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
			| <var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

Proceso de decodificación (Mapeo)

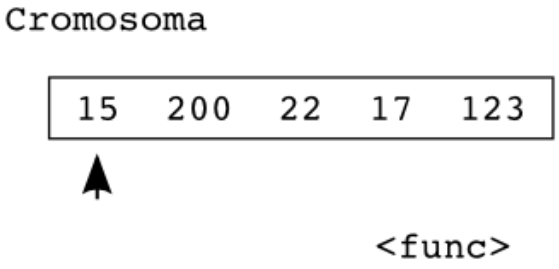
Cromosoma

15	200	22	17	123

<func>

I	<func>	::=	<expr> <op> <expr>
			| <preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
			| <var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

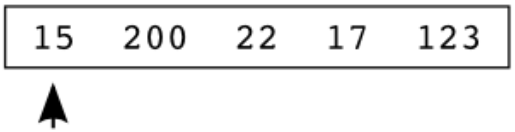
Proceso de decodificación (Mapeo)



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

Proceso de decodificación (Mapeo)

Cromosoma



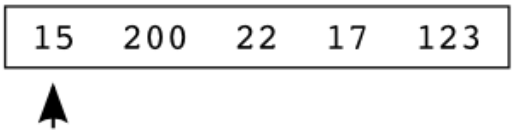
<func>

I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

15 mod 2 = 1

Proceso de decodificación (Mapeo)

Cromosoma

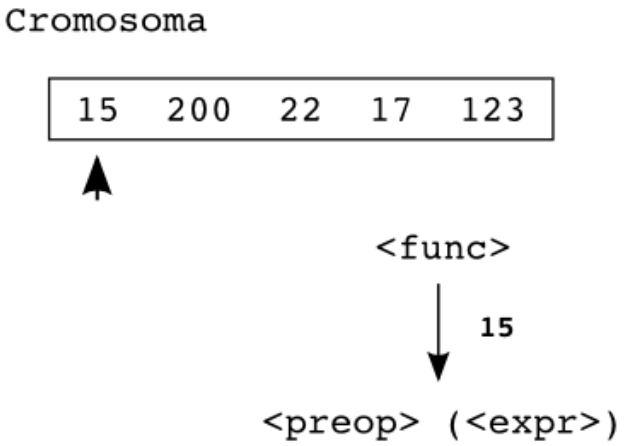


<func>

I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

15 mod 2 = 1

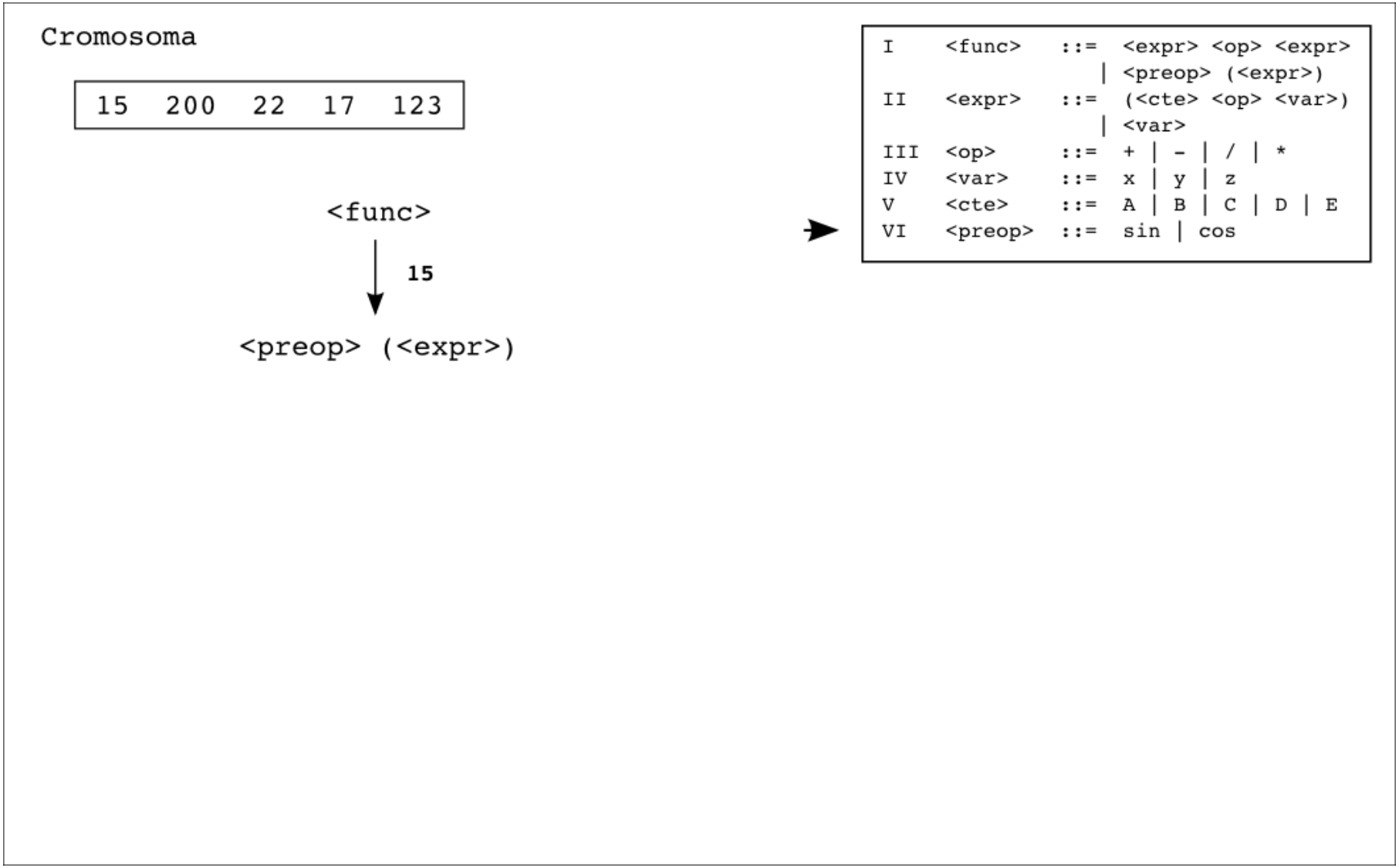
Proceso de decodificación (Mapeo)



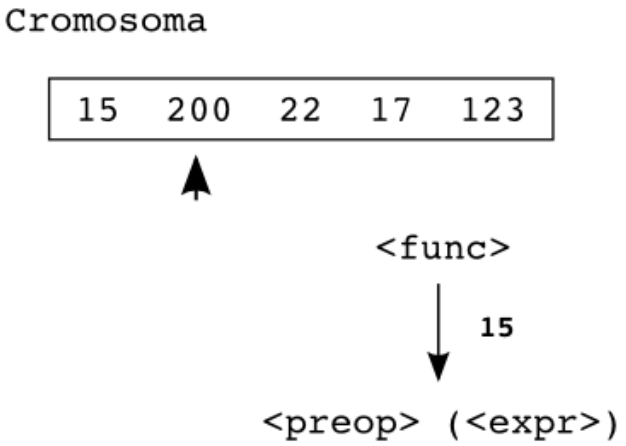
I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

15 mod 2 = 1

Proceso de decodificación (Mapeo)

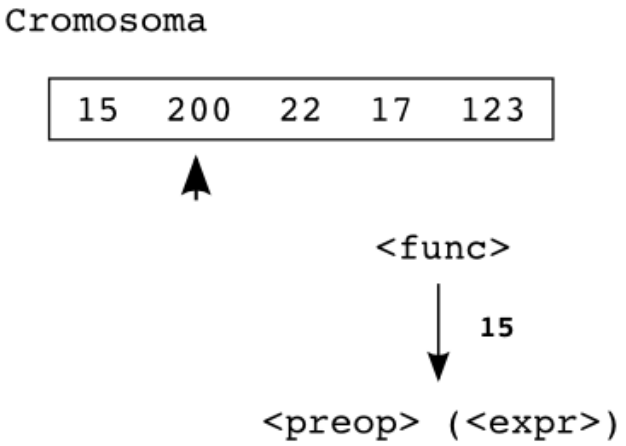


Proceso de decodificación (Mapeo)



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

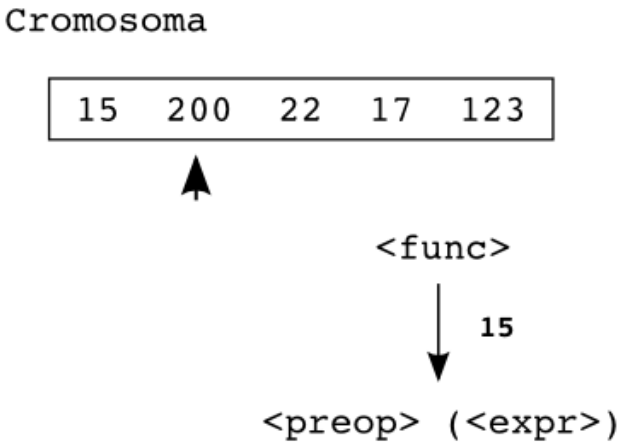
Proceso de decodificación (Mapeo)



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

200 mod 2 = 0

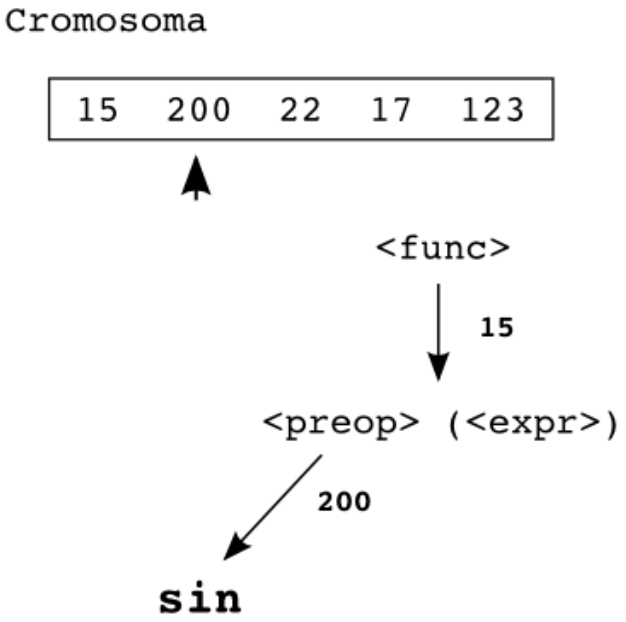
Proceso de decodificación (Mapeo)



I	<func>	::=	<expr> <op> <expr>
			| <preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
			| <var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

200 mod 2 = 0

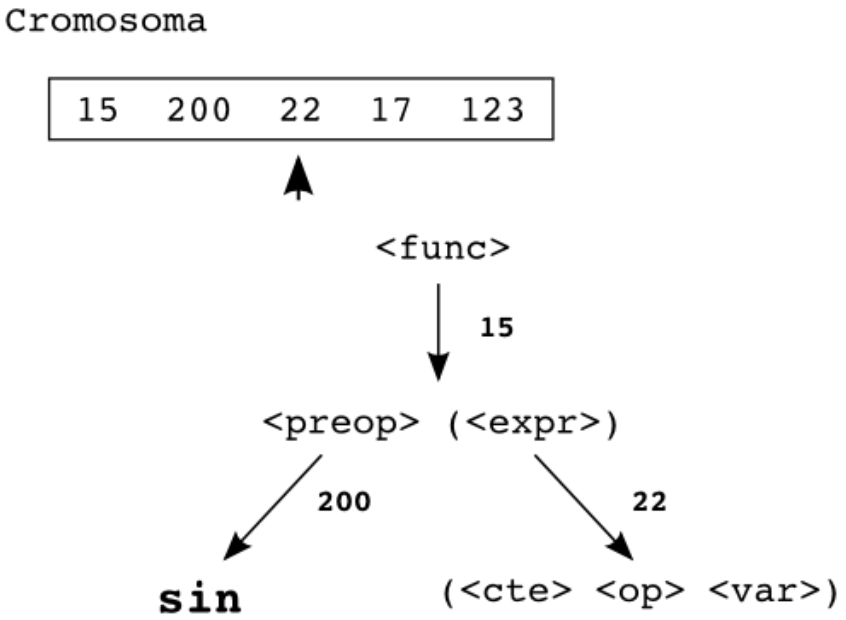
Proceso de decodificación (Mapeo)



I	<func>	::=	<expr> <op> <expr>
			| <preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
			| <var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

200 mod 2 = 0

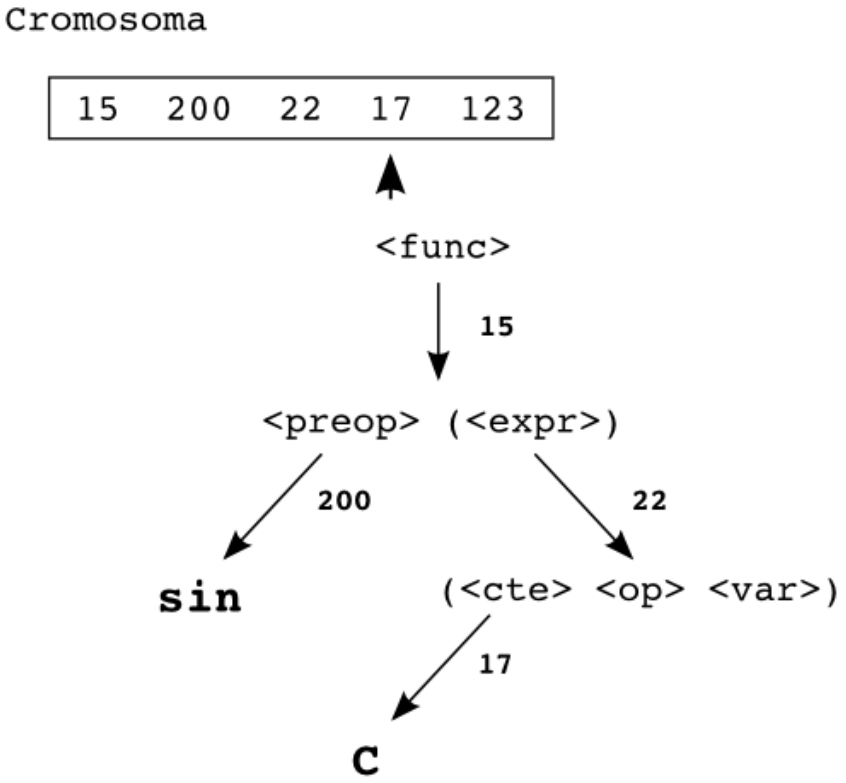
Proceso de decodificación (Mapeo)



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

22 mod 2 = 0

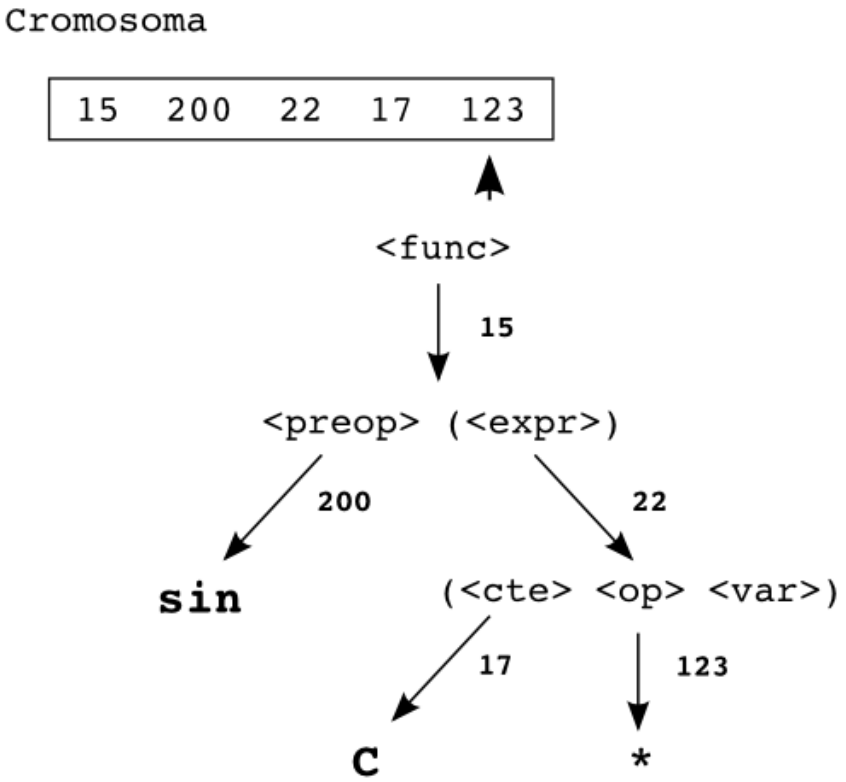
Proceso de decodificación (Mapeo)



I	<func>	::=	<expr> <op> <expr>
			| <preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
			| <var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

17 mod 5 = 2

Proceso de decodificación (Mapeo)



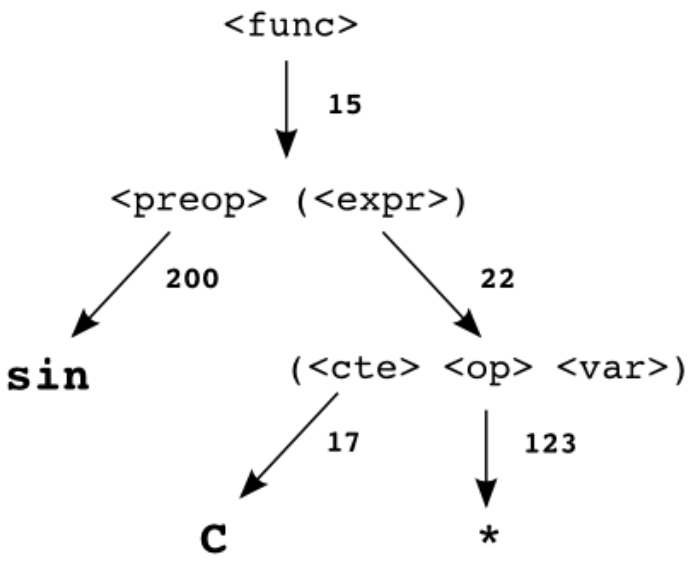
I	<func>	::=	<expr> <op> <expr>
			| <preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
			| <var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

123 mod 4 = 3

Proceso de decodificación (Mapeo)

Cromosoma

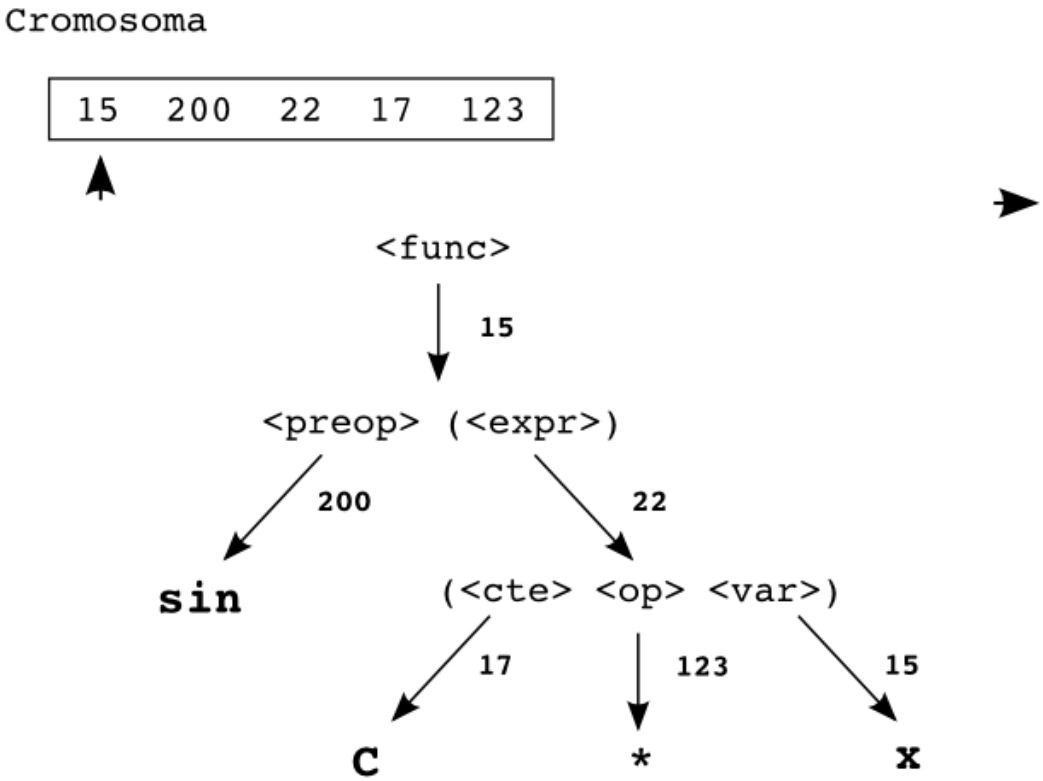
15	200	22	17	123



I	<func>	::=	<expr> <op> <expr>
			| <preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
			| <var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

WRAPS = 1

Proceso de decodificación (Mapeo)



I	<func>	::=	<expr> <op> <expr>
			| <preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
			| <var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

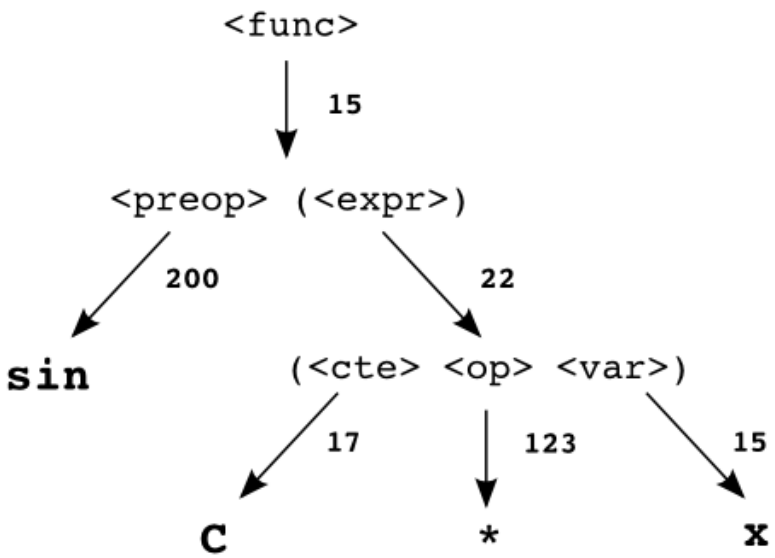
15 mod 3 = 0

WRAPS = 1

Proceso de decodificación (Mapeo)

Cromosoma

15	200	22	17	123



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

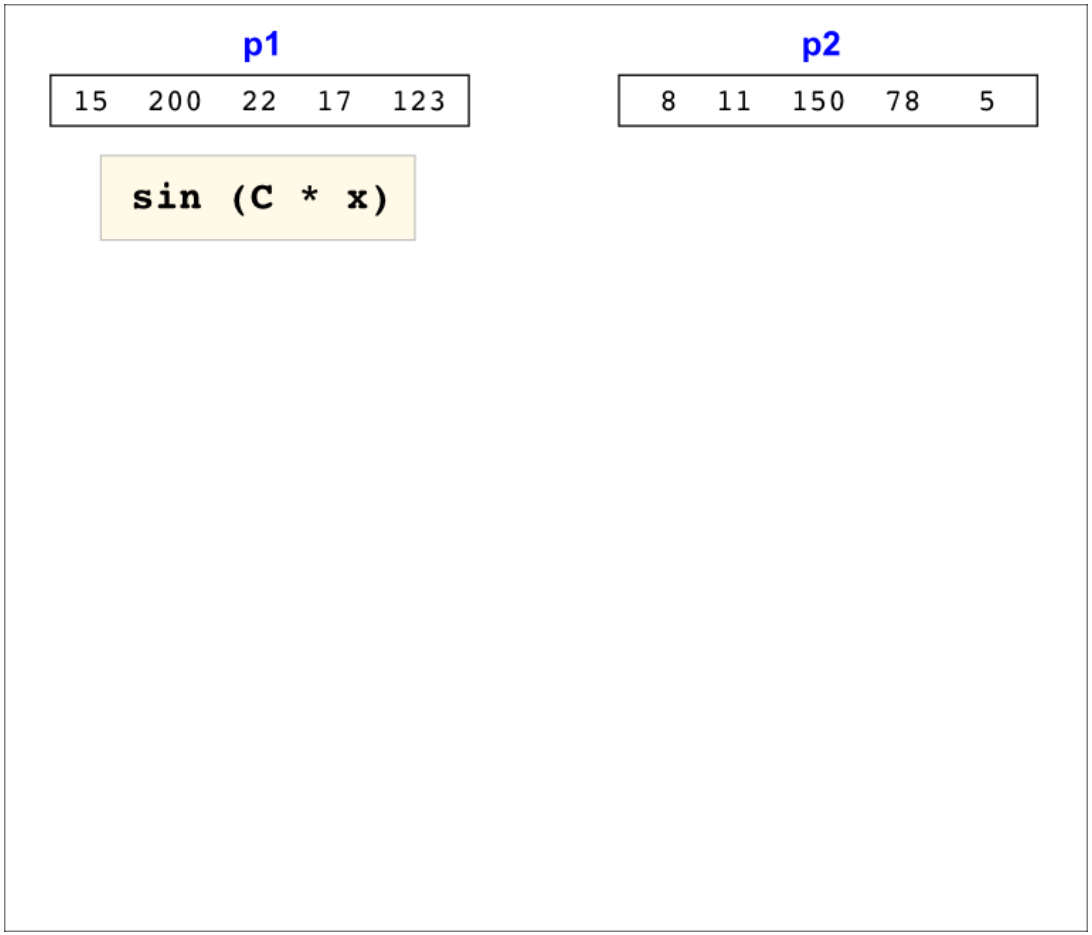
sin (C * x)

Operadores Genéticos

- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.

Operadores Genéticos

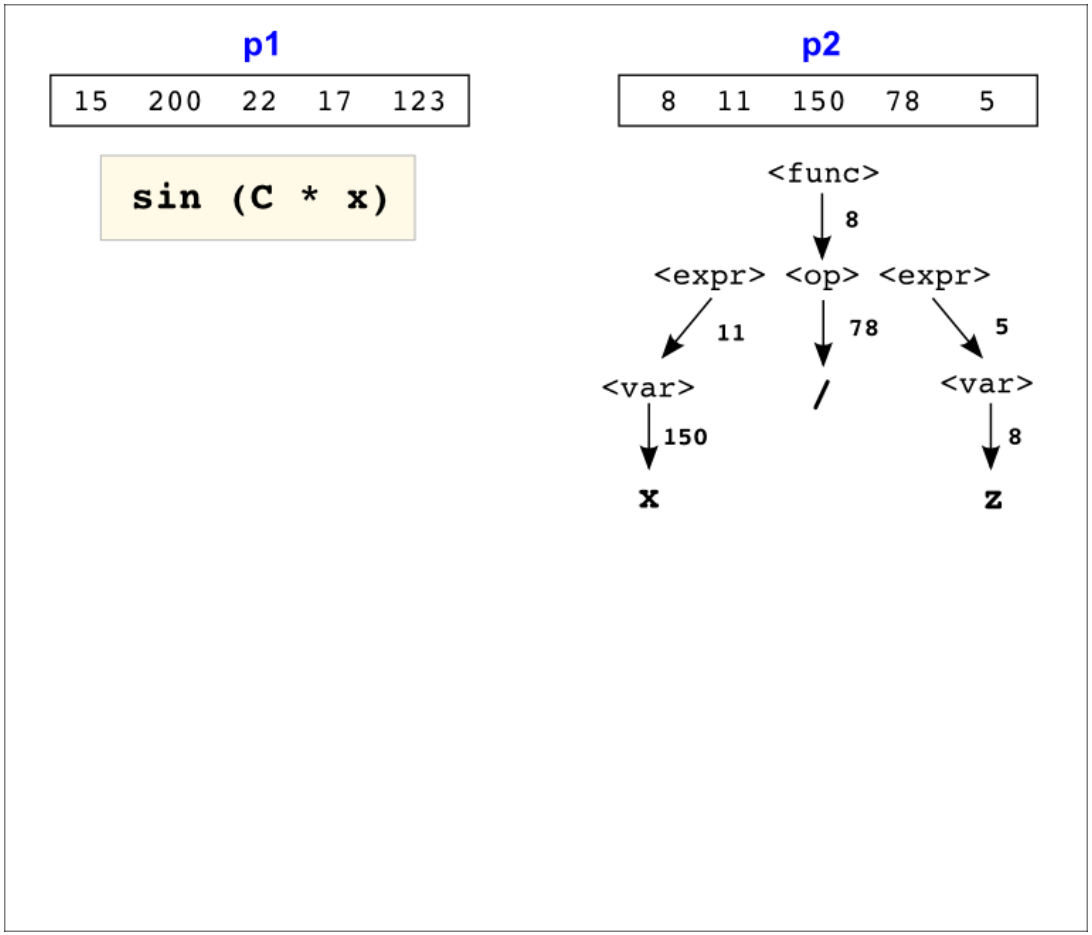
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I    <func>   ::=  <expr> <op> <expr>
                 | <preop> (<expr>)
II   <expr>   ::=  (<cte> <op> <var>)
                 | <var>
III  <op>     ::=  + | - | / | *
IV   <var>    ::=  x | y | z
V    <cte>    ::=  A | B | C | D | E
VI   <preop>  ::=  sin | cos
```

Operadores Genéticos

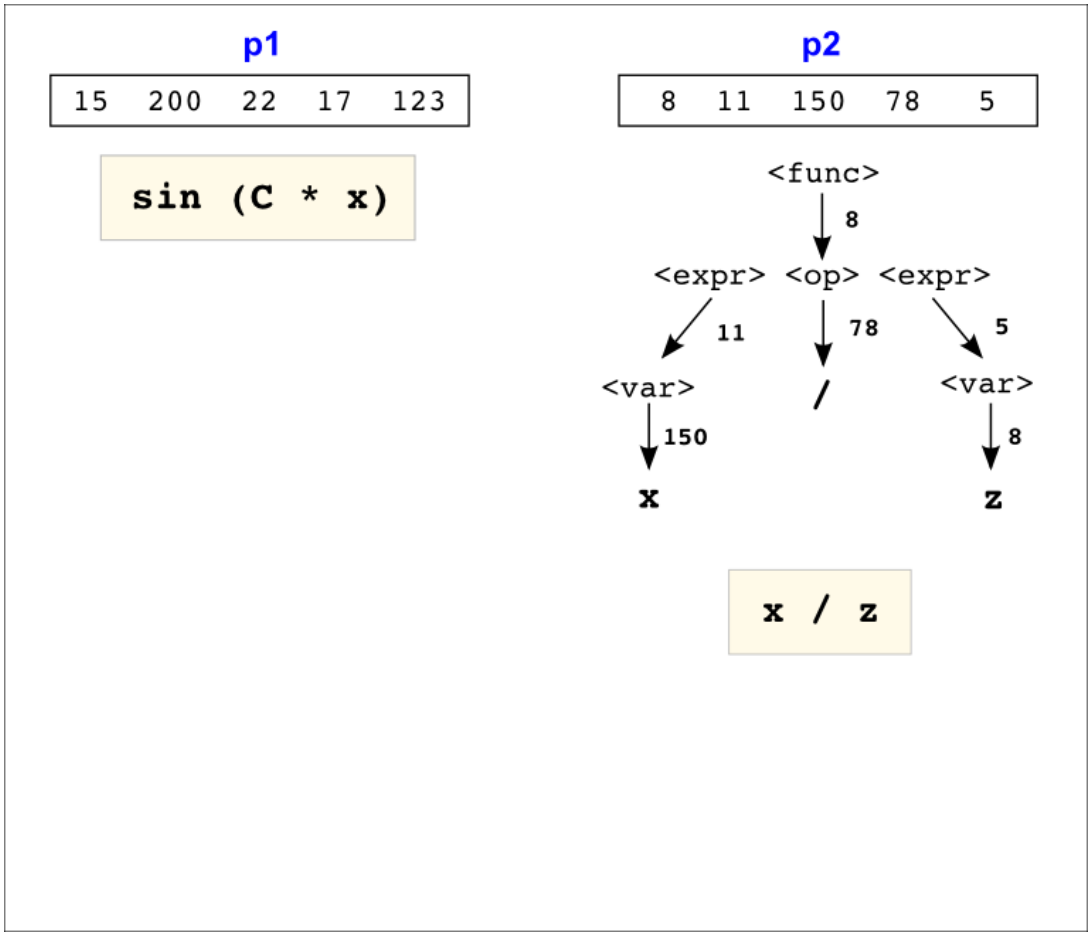
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

Operadores Genéticos

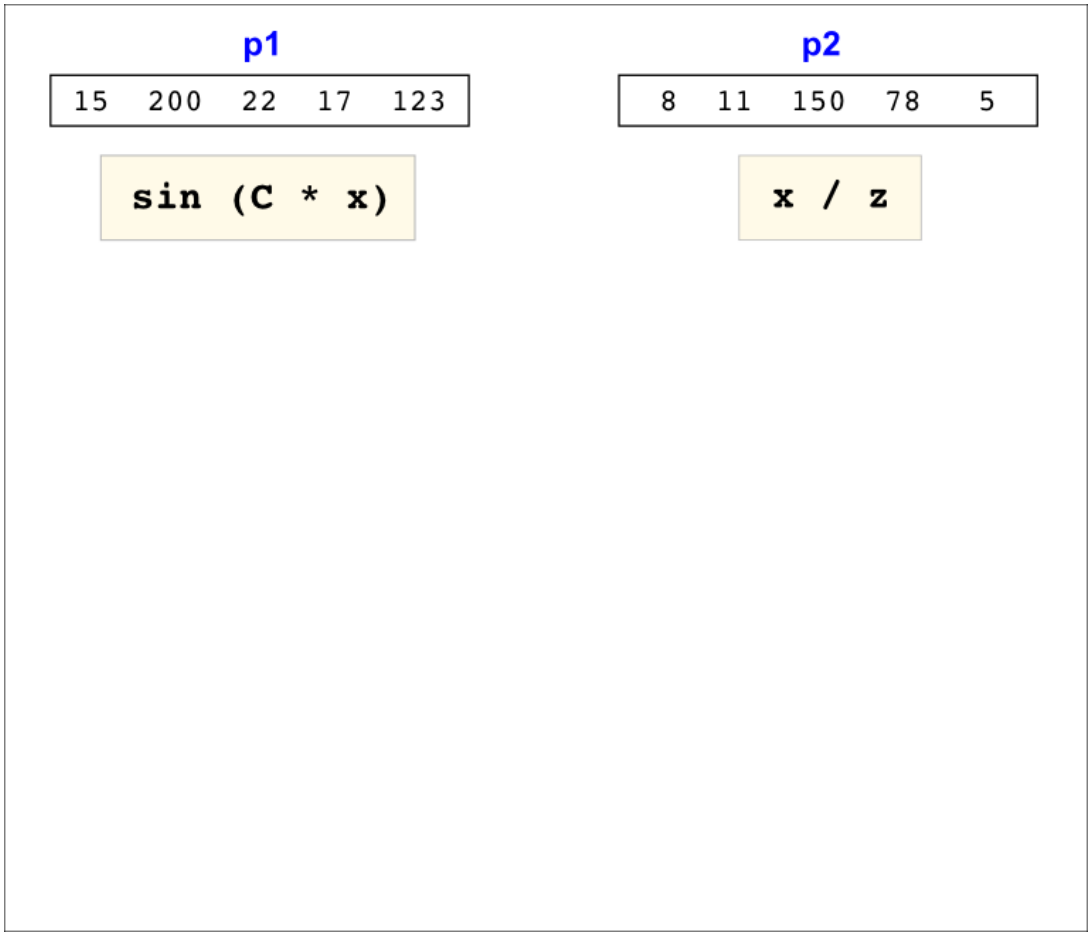
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

Operadores Genéticos

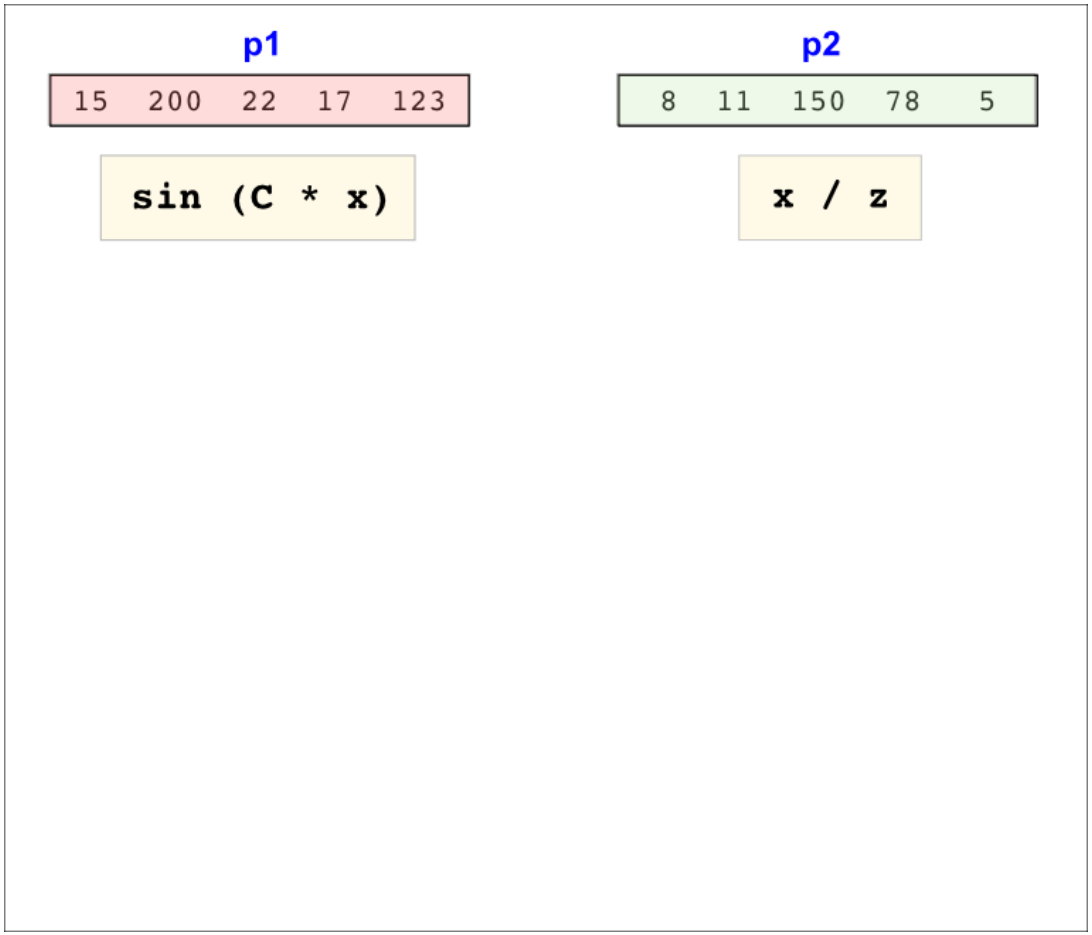
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I    <func>   ::=  <expr> <op> <expr>
                 | <preop> (<expr>)
II   <expr>   ::=  (<cte> <op> <var>)
                 | <var>
III  <op>     ::=  + | - | / | *
IV   <var>    ::=  x | y | z
V    <cte>    ::=  A | B | C | D | E
VI   <preop>  ::=  sin | cos
```

Operadores Genéticos

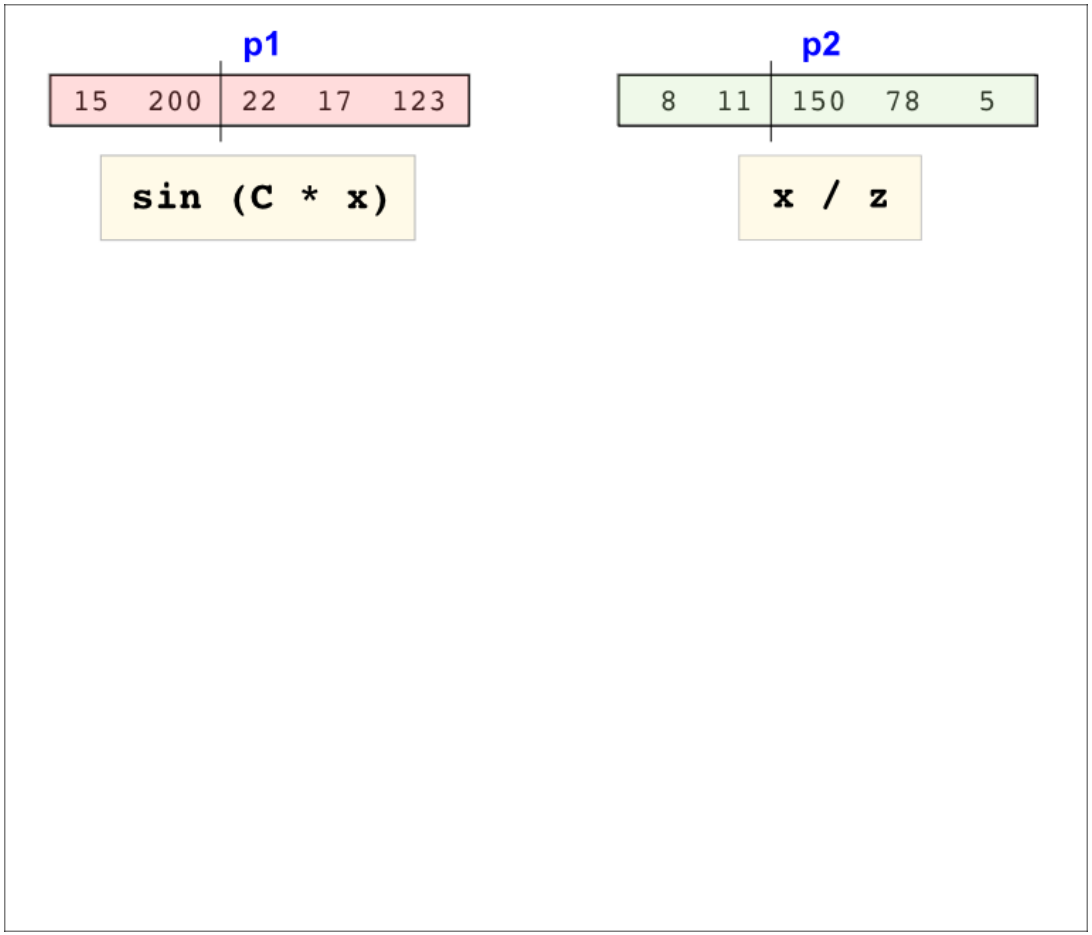
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I    <func>   ::=  <expr> <op> <expr>
                 | <preop> (<expr>)
II   <expr>   ::=  (<cte> <op> <var>)
                 | <var>
III  <op>     ::=  + | - | / | *
IV   <var>    ::=  x | y | z
V    <cte>    ::=  A | B | C | D | E
VI   <preop>  ::=  sin | cos
```

Operadores Genéticos

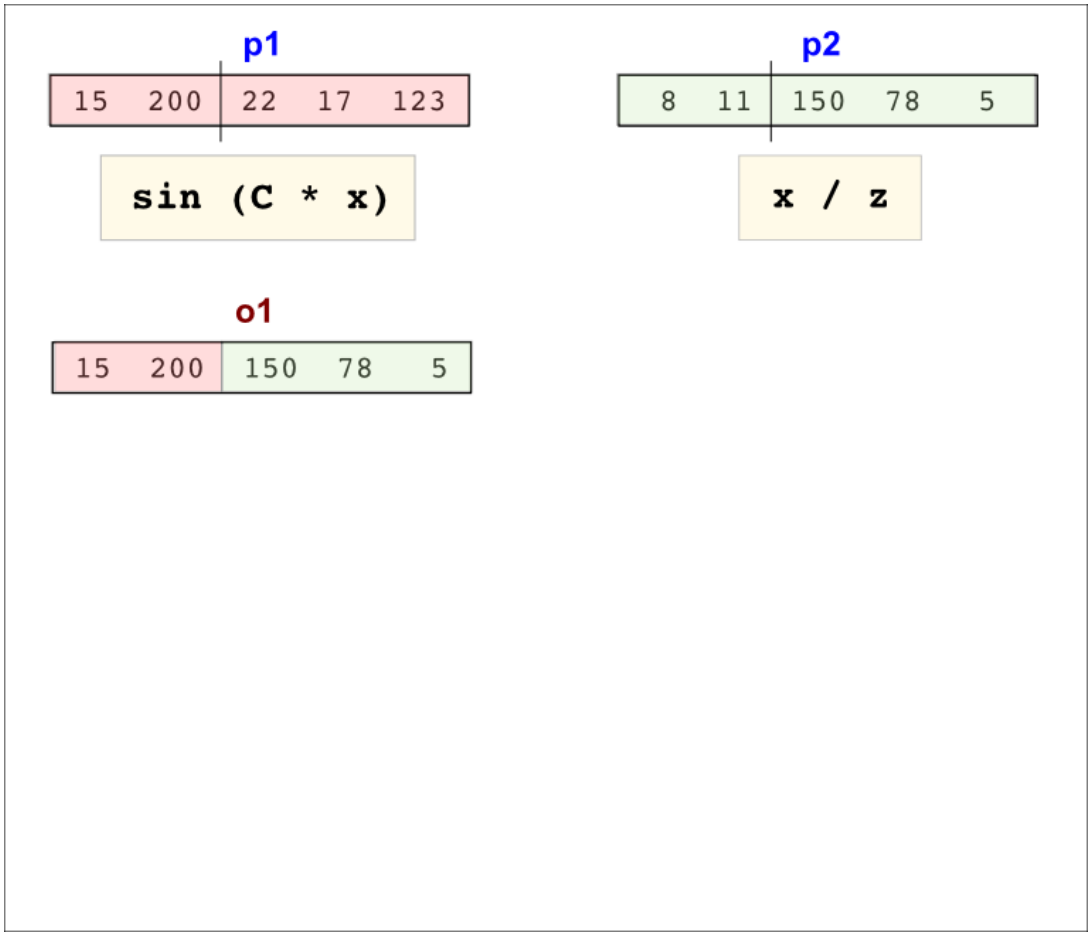
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I    <func>   ::=  <expr> <op> <expr>
                 | <preop> (<expr>)
II   <expr>   ::=  (<cte> <op> <var>)
                 | <var>
III  <op>     ::=  + | - | / | *
IV   <var>    ::=  x | y | z
V    <cte>    ::=  A | B | C | D | E
VI   <preop>  ::=  sin | cos
```

Operadores Genéticos

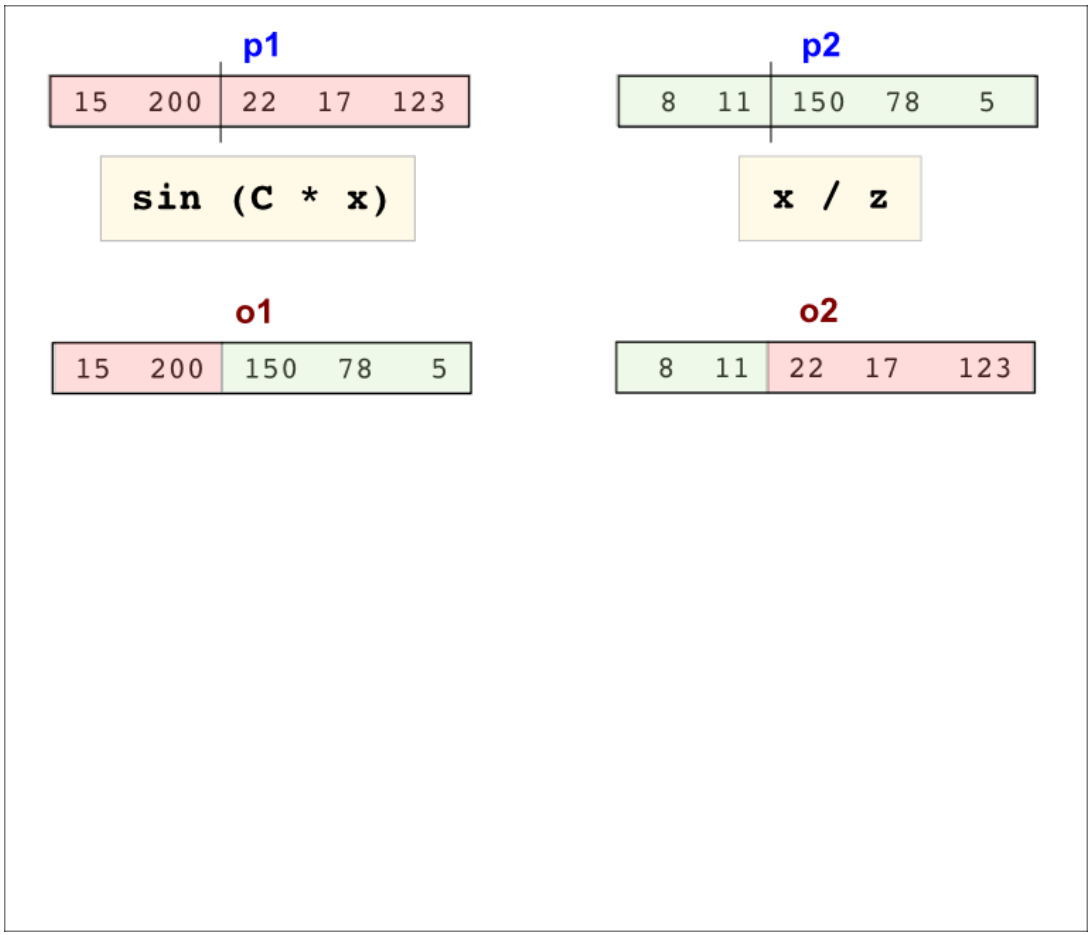
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I     <func>    ::=  <expr> <op> <expr>
                     | <preop> (<expr>)
II    <expr>    ::=  (<cte> <op> <var>)
                     | <var>
III   <op>      ::=  + | - | / | *
IV    <var>     ::=  x | y | z
V     <cte>     ::=  A | B | C | D | E
VI    <preop>   ::=  sin | cos
```

Operadores Genéticos

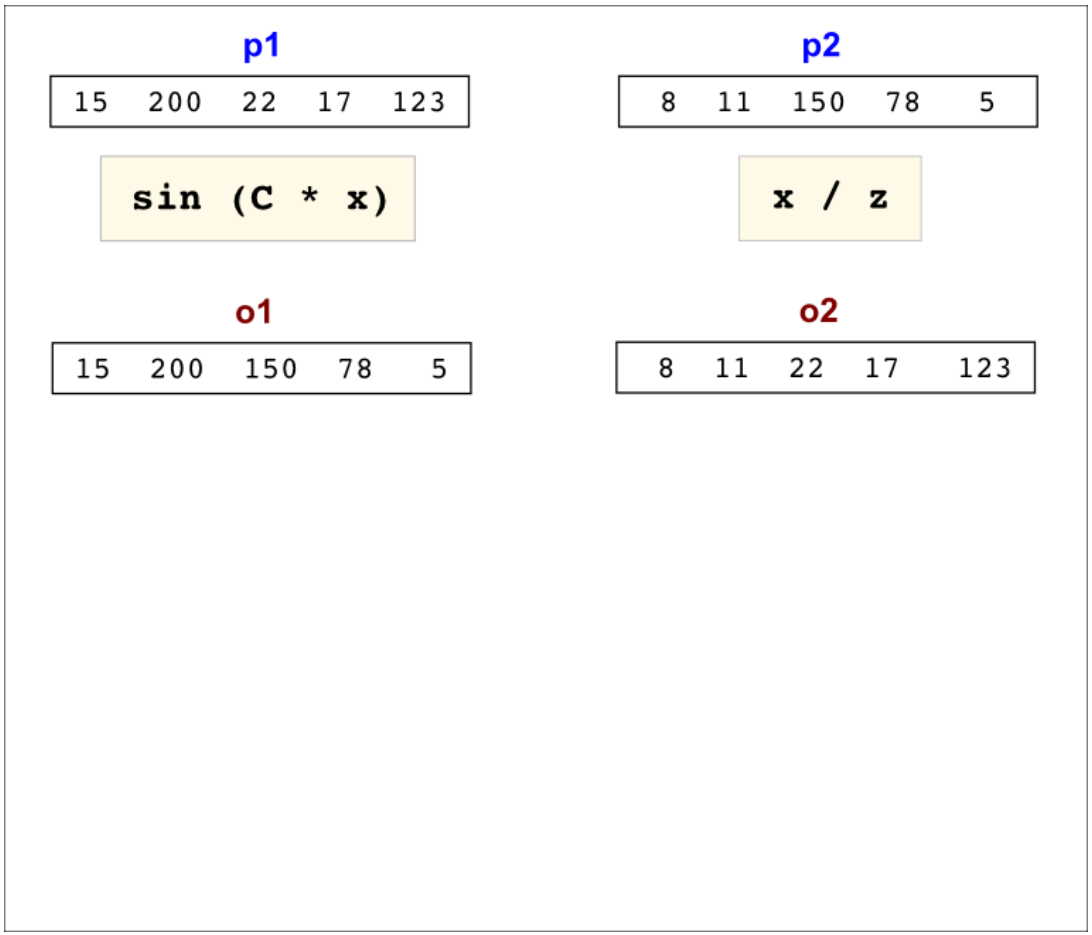
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I     <func>    ::=  <expr> <op> <expr>
                     | <preop> (<expr>)
II    <expr>    ::=  (<cte> <op> <var>)
                     | <var>
III   <op>      ::=  + | - | / | *
IV    <var>     ::=  x | y | z
V     <cte>     ::=  A | B | C | D | E
VI    <preop>   ::=  sin | cos
```

Operadores Genéticos

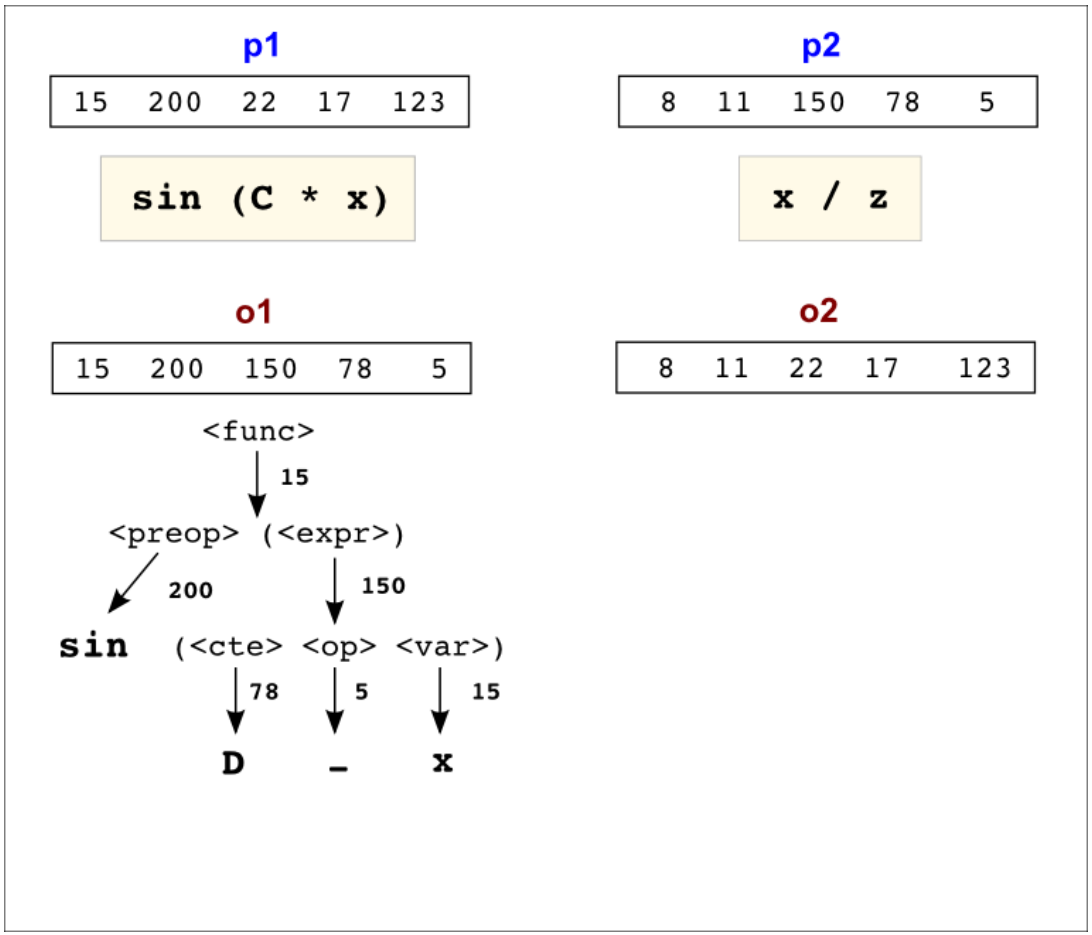
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I     <func>    ::=  <expr> <op> <expr>
                     | <preop> (<expr>)
II    <expr>    ::=  (<cte> <op> <var>)
                     | <var>
III   <op>      ::=  + | - | / | *
IV    <var>     ::=  x | y | z
V     <cte>     ::=  A | B | C | D | E
VI    <preop>   ::=  sin | cos
```

Operadores Genéticos

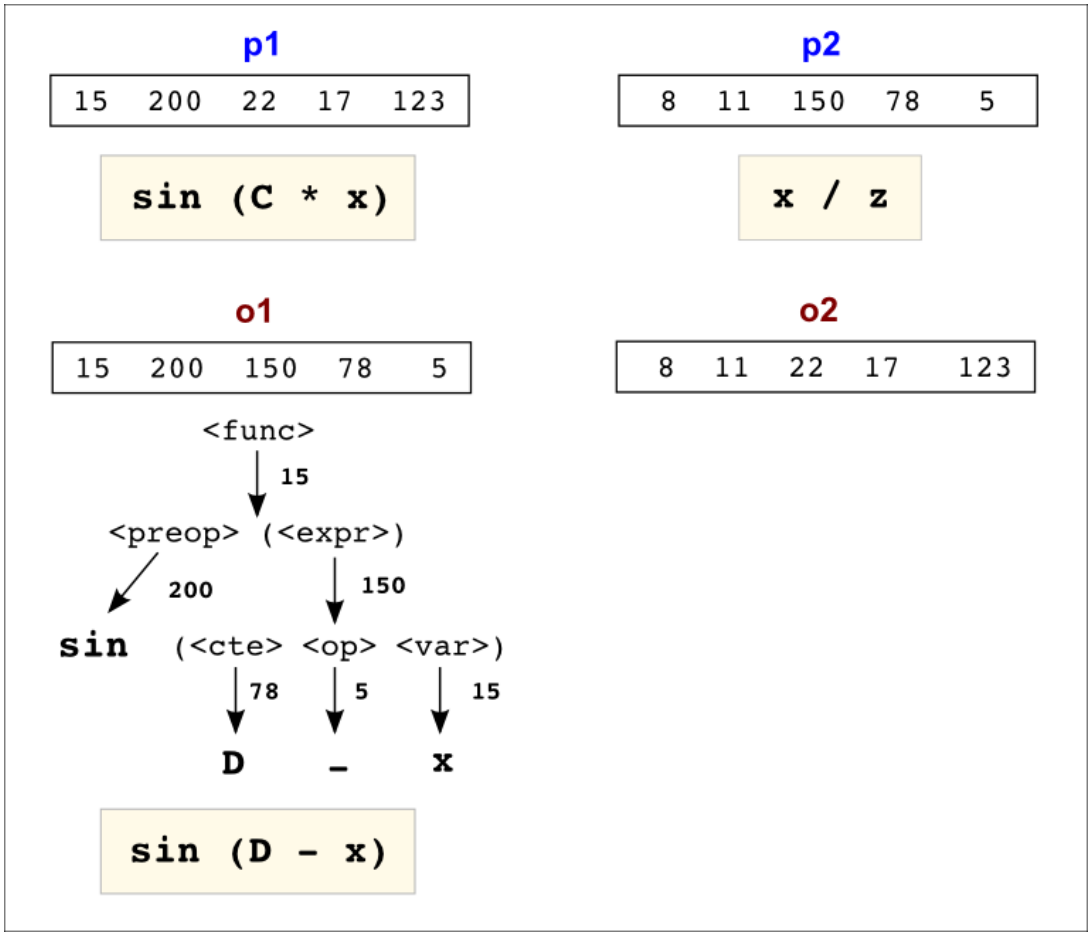
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I     <func>    ::=  <expr> <op> <expr>
                     | <preop> (<expr>)
II    <expr>    ::=  (<cte> <op> <var>)
                     | <var>
III   <op>      ::=  + | - | / | *
IV    <var>     ::=  x | y | z
V     <cte>     ::=  A | B | C | D | E
VI    <preop>   ::=  sin | cos
```

Operadores Genéticos

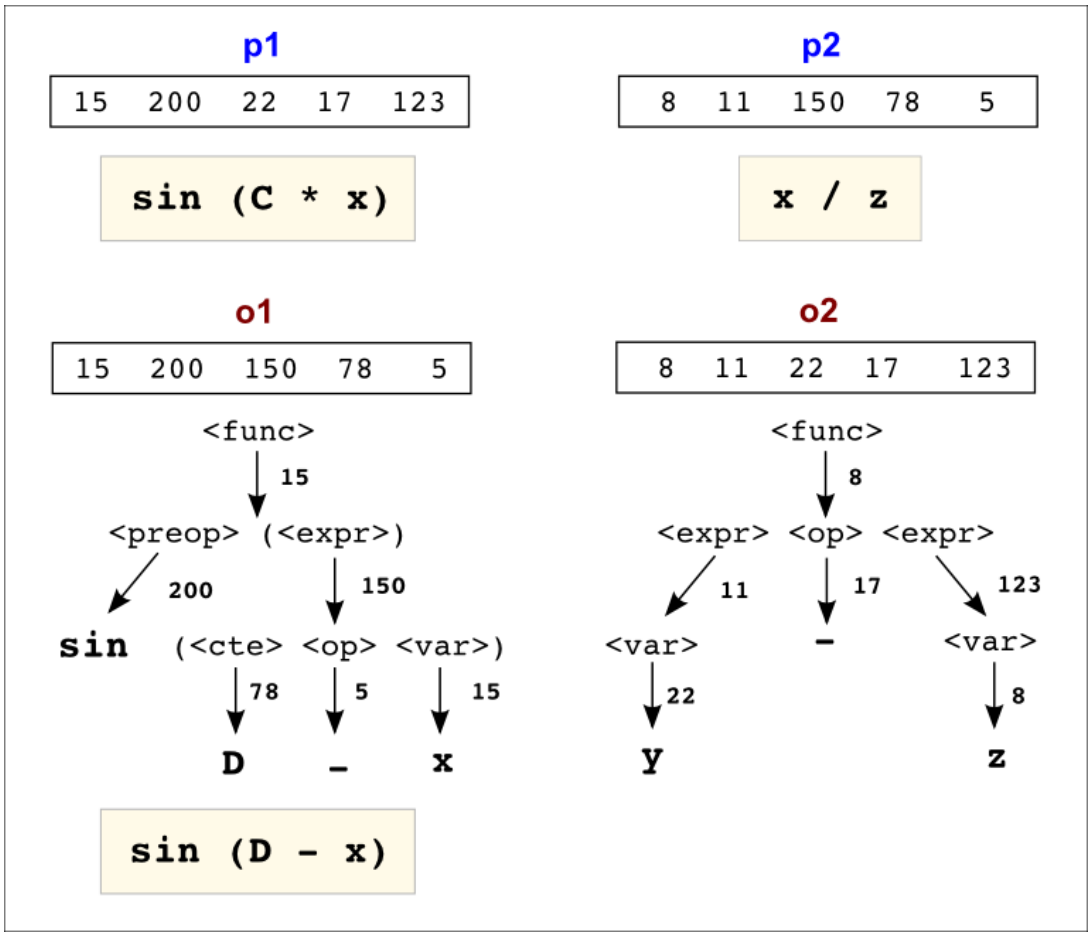
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I    <func>    ::=  <expr> <op> <expr>
                  | <preop> (<expr>)
II   <expr>    ::=  (<cte> <op> <var>)
                  | <var>
III  <op>      ::=  + | - | / | *
IV   <var>     ::=  x | y | z
V    <cte>     ::=  A | B | C | D | E
VI   <preop>   ::=  sin | cos
```

Operadores Genéticos

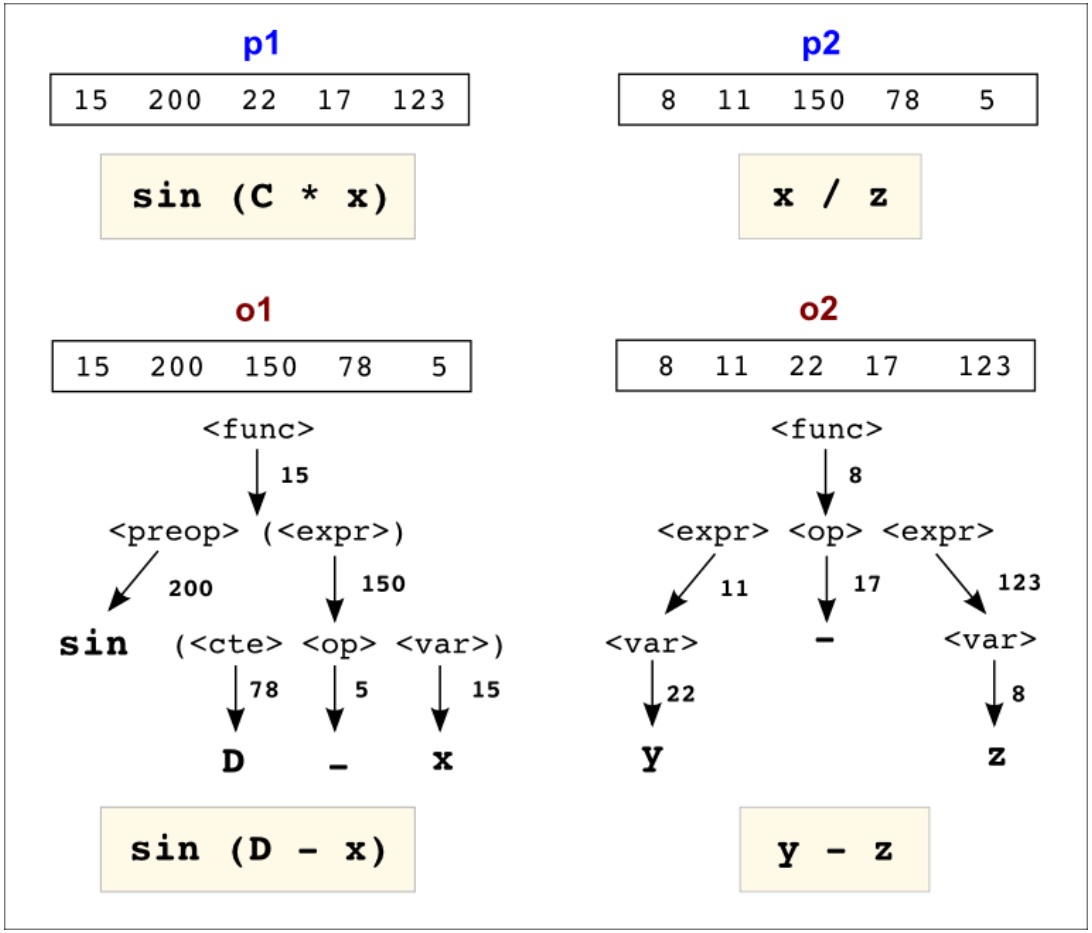
- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I     <func>    ::=  <expr> <op> <expr>
                     | <preop> (<expr>)
II    <expr>    ::=  (<cte> <op> <var>)
                     | <var>
III   <op>      ::=  + | - | / | *
IV    <var>     ::=  x | y | z
V     <cte>     ::=  A | B | C | D | E
VI    <preop>   ::=  sin | cos
```

Operadores Genéticos

- Los operadores genéticos **se aplican al genotipo de manera sencilla**.
- El **cruce** no lleva necesariamente a una combinación exacta de fenotipos.




```
I     <func>    ::=  <expr> <op> <expr>
                     | <preop> (<expr>)
II    <expr>    ::=  (<cte> <op> <var>)
                     | <var>
III   <op>      ::=  + | - | / | *
IV    <var>     ::=  x | y | z
V     <cte>     ::=  A | B | C | D | E
VI    <preop>   ::=  sin | cos
```

Operadores Genéticos

- La **mutación** es más "radical" si afecta a un símbolo no terminal.
- Es posible que la mutación no surta efecto si el nuevo valor tiene el mismo módulo sobre la regla a decodificar.
  - Cuanto mayor es el número de producciones de una regla, menor es la probabilidad de cada elemento.

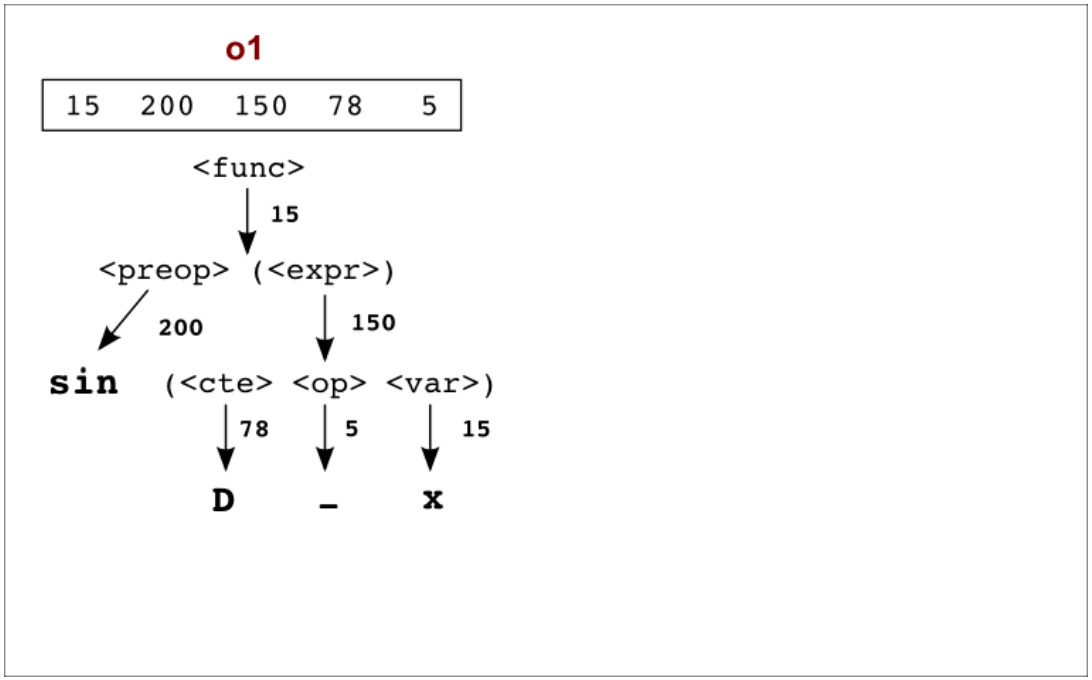
o1

15	200	150	78	5

```
I    <func>    ::=  <expr> <op> <expr>
                    | <preop> (<expr>)
II   <expr>    ::=  (<cte> <op> <var>)
                    | <var>
III  <op>      ::=  + | - | / | *
IV   <var>     ::=  x | y | z
V    <cte>     ::=  A | B | C | D | E
VI   <preop>   ::=  sin | cos
```

Operadores Genéticos

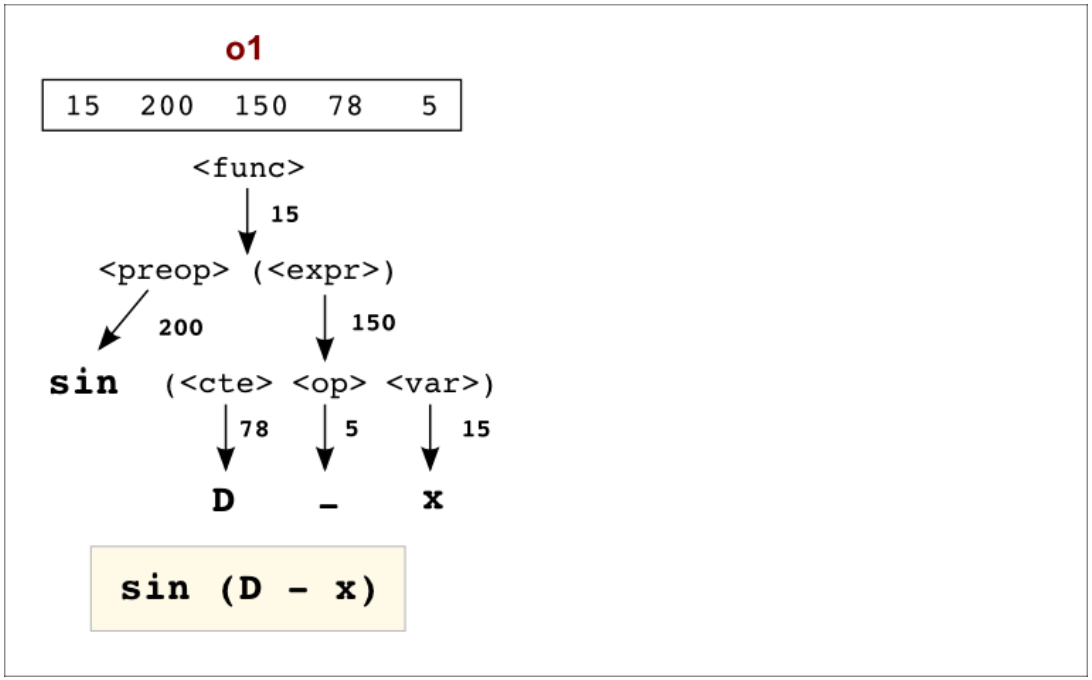
- La **mutación** es más "radical" si afecta a un símbolo no terminal.
- Es posible que la mutación no surta efecto si el nuevo valor tiene el mismo módulo sobre la regla a decodificar.
  - Cuanto mayor es el número de producciones de una regla, menor es la probabilidad de cada elemento.




```
I    <func>    ::=  <expr> <op> <expr>
                  | <preop> (<expr>)
II   <expr>    ::=  (<cte> <op> <var>)
                  | <var>
III  <op>      ::=  + | - | / | *
IV   <var>     ::=  x | y | z
V    <cte>     ::=  A | B | C | D | E
VI   <preop>   ::=  sin | cos
```

Operadores Genéticos

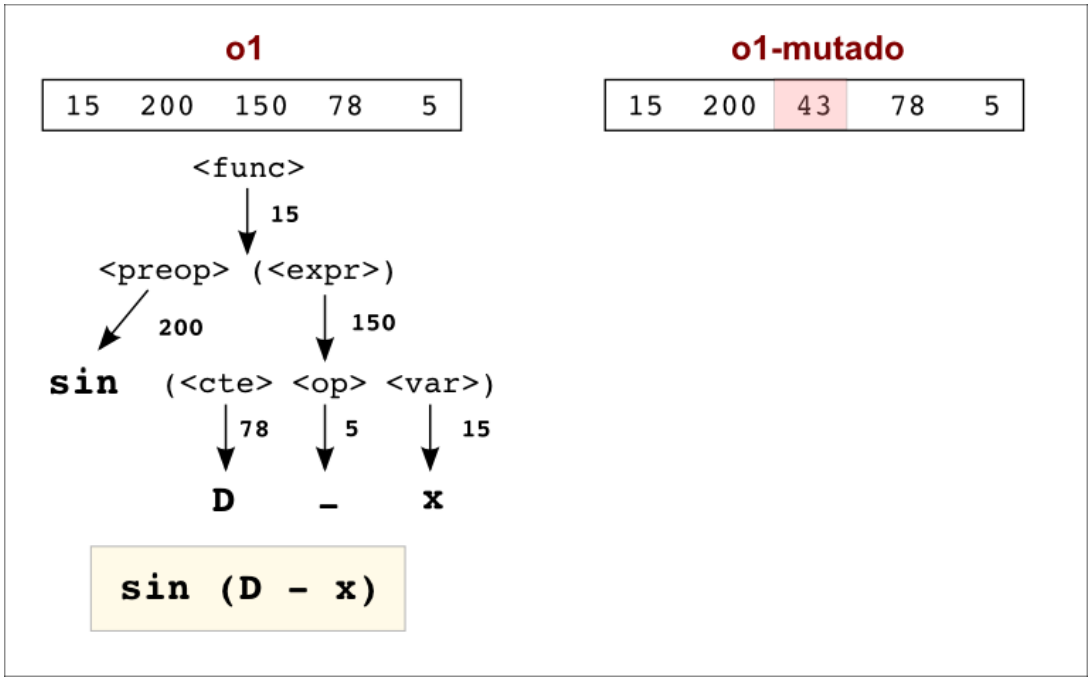
- La **mutación** es más "radical" si afecta a un símbolo no terminal.
- Es posible que la mutación no surta efecto si el nuevo valor tiene el mismo módulo sobre la regla a decodificar.
  - Cuanto mayor es el número de producciones de una regla, menor es la probabilidad de cada elemento.




```
I    <func>    ::=  <expr> <op> <expr>
                  | <preop> (<expr>)
II   <expr>    ::=  (<cte> <op> <var>)
                  | <var>
III  <op>      ::=  + | - | / | *
IV   <var>     ::=  x | y | z
V    <cte>     ::=  A | B | C | D | E
VI   <preop>   ::=  sin | cos
```

Operadores Genéticos

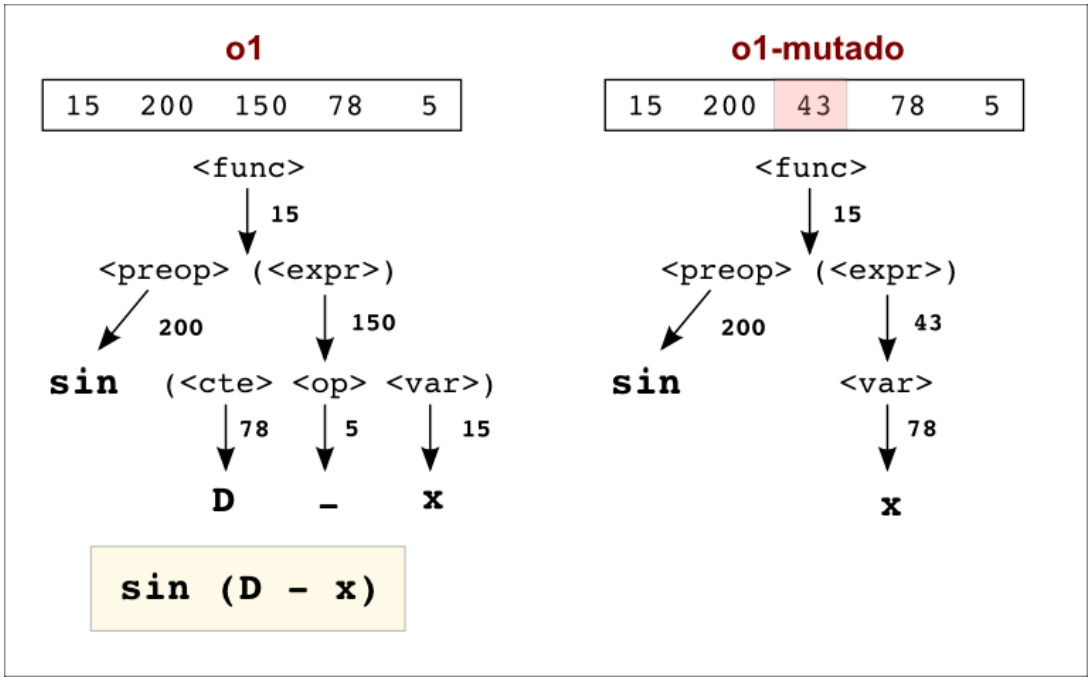
- La **mutación** es más "radical" si afecta a un símbolo no terminal.
- Es posible que la mutación no surta efecto si el nuevo valor tiene el mismo módulo sobre la regla a decodificar.
  - Cuanto mayor es el número de producciones de una regla, menor es la probabilidad de cada elemento.




```
I     <func>    ::=  <expr> <op> <expr>
                     | <preop> (<expr>)
II    <expr>    ::=  (<cte> <op> <var>)
                     | <var>
III   <op>      ::=  + | - | / | *
IV    <var>     ::=  x | y | z
V     <cte>     ::=  A | B | C | D | E
VI    <preop>   ::=  sin | cos
```

Operadores Genéticos

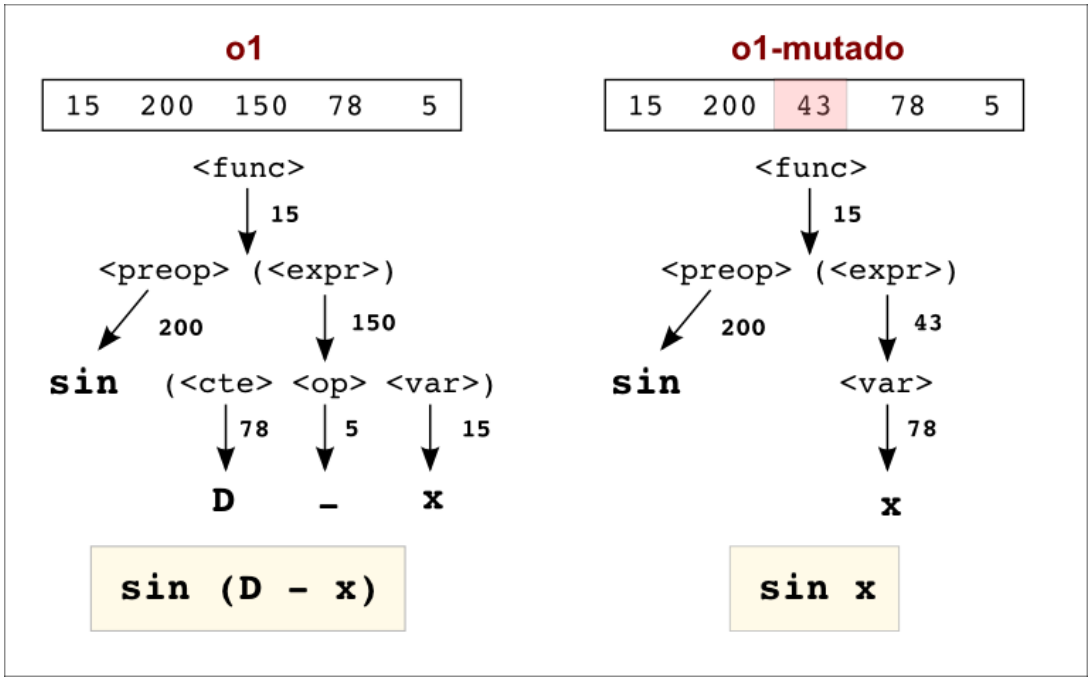
- La **mutación** es más "radical" si afecta a un símbolo no terminal.
- Es posible que la mutación no surta efecto si el nuevo valor tiene el mismo módulo sobre la regla a decodificar.
  - Cuanto mayor es el número de producciones de una regla, menor es la probabilidad de cada elemento.



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

Operadores Genéticos

- La **mutación** es más "radical" si afecta a un símbolo no terminal.
- Es posible que la mutación no surta efecto si el nuevo valor tiene el mismo módulo sobre la regla a decodificar.
  - Cuanto mayor es el número de producciones de una regla, menor es la probabilidad de cada elemento.



I	<func>	::=	<expr> <op> <expr>
		|	<preop> (<expr>)
II	<expr>	::=	(<cte> <op> <var>)
		|	<var>
III	<op>	::=	+ | - | / | *
IV	<var>	::=	x | y | z
V	<cte>	::=	A | B | C | D | E
VI	<preop>	::=	sin | cos

Beneficios de la Evolución Gramatical

Ventajas sobre Algoritmos Genéticos

- Utiliza el cromosoma parcial o totalmente (**tamaño variable**).
- La **gramática permite ajustar la búsqueda** sin tocar código del motor de optimización.
  - Incorpora restricciones.
  - Delimita los individuos no válidos.

Ventajas sobre Programación Genética

- La **codificación** de los árboles es **sencilla** en los cromosomas.
- **Controla el bloating** a través de la limitación de los *wraps*.
- Los **operadores genéticos** son más **sencillos de aplicar** en genotipo.

Inconvenientes de Evolución Gramatical

- Puede ser complicado ajustar la gramática.
- El descarte de individuos no válidos puede ser lento si se permiten muchos *wraps*.

Aplicaciones

- Obtención de **modelos para glucosa** en diabéticos.
- Diseño automático de **gestores de memoria dinámica** (*DMMs*).
- Diseño automático de **configuraciones de cache**.
- Creación automática de **máquinas de estados finitas** (*FSMs*).
- Construcción de reglas en **sistemas de trading**.
- Obtención de modelos en **entornos dinámicos**.
- Problemas de regresión simbólica.
- Algoritmos de comportamiento en robots.
- Evolución de reglas para reconocimiento de sonidos y palabras.
- Evolución de curvas fractales de alta dimensionalidad.

```
N = {func, exprgluc, gluc, exprch, varch, exprins, varins, op, preop, idx,
    cte, dgt}
T = { +,-,*,/, sin, cos, tan, exp, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, GL, CH,
    IS, IL, K}
S = {func}
P = {I, II ,III ,IV ,V ,VI ,VII, VIII, IX, X, XI, XII}

I    <func> ::= <exprgluc> <op> <exprch> <op> <exprins>

II   <exprgluc> ::= <preop> (<gluc>)
        |(<cte> <op> <gluc>)
        |<gluc>

III  <gluc> ::= #{GL[k_<idx>]}|#{K}

IV   <exprch> ::= <exprch> <op> <exprch>
        |<preop> (<exprch>)
        |<varch>

V    <varch> ::= #{CH[k_<idx>]}|#{K}|<cte>

VI   <exprins> ::= <exprins> <op> <exprins>
        |<preop> (<exprins>)
        |<varins>

VII  <varins> ::= #{IS[k_<idx>]}|#{IL[k_<idx>]}|#{K}|<cte>

VIII <op> ::=+|-|/|*
IX   <preop>::=sin|cos|tan|exp
X    <idx> ::= <dgt><dgt>
XI   <cte> ::= <dgt><dgt>.<dgt><dgt>
XII  <dgt>::=0|1|2|3|4|5|6|7|8|9
```

Aplicaciones

- Obtención de **modelos para glucosa** en diabéticos.
- Diseño automático de **gestores de memoria dinámica** (*DMMs*).
- Diseño automático de **configuraciones de cache**.
- Creación automática de **máquinas de estados finitas** (*FSMs*).
- Construcción de reglas en **sistemas de trading**.

- Obtención de modelos en **entornos dinámicos**.
- Problemas de regresión simbólica.
- Algoritmos de comportamiento en robots.
- Evolución de reglas para reconocimiento de sonidos y palabras.
- Evolución de curvas fractales de alta dimensionalidad.

```
N = {<DynamicMemoryManager>, <Allocators>,
     <AllocatorSize>, <AllocatorMaxSize>,
     <AllocatorClass>, <Size>, <MaxSize>,
     <AllowSplitting>, <AllowCoalescing>,
     <DataStructure>, <AllocationMechanism>,
     <AllocationPolicy>}
T = {SegregatedFreeList, SimpleSegregatedStorage, ExactSegregatedFit, BuddySystemBinary,
     BuddySystemFibonacci, 4, 7, 8, {block sizes ommited here for the sake of space)... , 7832240, true,
     false, SLL, DLL, BTREE, FIRST, BEST, EXACT, FIFO, LIFO}
S = <DynamicMemoryManager>
I    <DynamicMemoryManager> ::= <Allocators>
II   <Allocators>           ::= <AllocatorSize>
                             |<AllocatorMaxSize>
III  <AllocatorSize>        ::= <AllocatorClass>
                                 <AllowSplitting>
                                 <AllowCoalescing>
                                 <Size>
                                 <DataStructure>
                                 <AllocationMechanism>
                                 <AllocationPolicy>
                                 <Allocators>
IV   <AllocatorMaxSize>     ::= <AllocatorClass>
                                 <AllowSplitting>
                                 <AllowCoalescing>
                                 <MaxSize>
                                 <DataStructure>
                                 <AllocationMechanism>
                                 <AllocationPolicy>
V    <AllocatorClass>    ::= SegregatedFreeList
                            |SimpleSegregatedStorage
                            |ExactSegregatedFit
                            |BuddySystemBinary
                            |BuddySystemFibonacci
VI   <Size>                ::= 4|7|8|12|14|15|16|17|18|19|20|21|23|24|25|26|28|31|32|36|40|44
                            |48|52|56|60|64|72|76|79|80|84|88|92|96|100|104|108|112|116|120|128
                            |132|136|140|144|152|160|168|172|176|180|184|188|192|200|202|208|212
                            |216|220|224|228|232|236|240|248|252|256|264|276|280|284|288|300|303
                            312|316|320|324|336|348|352|356|360|380|384|396|400|424|432|440|444
                            |448|457|480|492|496|500|504|512|525|528|536|540|556|564|576|584|600
                            |608|617|624|626|630|631|632|648|660|672|692|713|720|745|756|768|772
                            |784|800|808|820|...|820840|995732|1011432|1034632|1274436|1295112
                            |1296000|1616064|1916188|1947512|1991464|2548872|2760556|2804912
                            |3832376|3916120|3979304|5521112
VII  <MaxSize>             ::= 7832240
VIII <AllowSplitting>      ::= true|false
IX   <AllowCoalescing>     ::= true|false
X    <DataStructure>       ::= SLL|DLL|BTREE
XI   <AllocationMechanism> ::= FIRST|BEST|EXACT
XII  <AllocationPolicy>    ::= FIFO|LIFO
```

Aplicaciones

- Obtención de **modelos para glucosa** en diabéticos.
- Diseño automático de **gestores de memoria dinámica** (*DMMs*).
- Diseño automático de **configuraciones de cache**.
- Creación automática de **máquinas de estados finitas** (*FSMs*).
- Construcción de reglas en **sistemas de trading**.

- Obtención de modelos en **entornos dinámicos**.
- Problemas de regresión simbólica.
- Algoritmos de comportamiento en robots.
- Evolución de reglas para reconocimiento de sonidos y palabras.
- Evolución de curvas fractales de alta dimensionalidad.

```
N = { <DineroParams>, <CacheSizeB>, <LineSizeB>,
      <ReplAlg>, <Assoc>, <PrefAlg>, <WritePol> }
T = { 1, 2, 4, 8, 16, 32, 64, 128 512, 1024, 2048,
      4096, 8192, 16384, 32768, 65536, l, f, r, m,
      d, a, n }
S = <DineroParams>

P = { I    <DineroParams> ::= -l1-isize <CacheSizeB>
                              -l1-ibsize <LineSizeB>
                              -l1-irepl <ReplAlg>
                              -l1-iassoc <Assoc>
                              -l1-ifetch <PrefAlg>
                              -l1-dsize <CacheSizeB>
                              -l1-dbsize <LineSizeB>
                              -l1-drepl <ReplAlg>
                              -l1-dassoc <Assoc>
                              -l1-dfetch <PrefAlg>
                              -l1-dwback <WritePol>

      II    <CacheSizeB> ::= 512 | 1024 | 2048 | 4096
                           | 8192 | 16384 | 32768
                           | 65536
      III   <LineSizeB> ::= 8 | 16 | 32 | 64
      IV    <ReplAlg> ::= l | f | r
      V     <Assoc> ::= 1 | 2 | 4 | 8 | 16 | 32 | 64
                      | 128
      VI    <PrefAlg> ::= m | d | a
      VII    <WritePol> ::= a | n  }
```

Aplicaciones

- Obtención de **modelos para glucosa** en diabéticos.
- Diseño automático de **gestores de memoria dinámica** (*DMMs*).
- Diseño automático de **configuraciones de cache**.
- Creación automática de **máquinas de estados finitas** (*FSMs*).
- Construcción de reglas en **sistemas de trading**.
- Obtención de modelos en **entornos dinámicos**.
- Problemas de regresión simbólica.
- Algoritmos de comportamiento en robots.
- Evolución de reglas para reconocimiento de sonidos y palabras.
- Evolución de curvas fractales de alta dimensionalidad.

Gramática para máquina de estados

```
N = {<FSM>,<State>,<StateId>,<Output>,<Trans>}
T = {Reset,0,1,A,B,C,D}
S = <FSM>

I    <FSM>      ::= <State>
II   <State>    ::= <StateId><Output><Trans><Trans>
III  <Trans>    ::= <State> | Reset
IV   <Output>   ::= 0 | 1
V    <StateId>  ::= A | B | C | D
```

Aplicaciones

- Obtención de **modelos para glucosa** en diabéticos.
- Diseño automático de **gestores de memoria dinámica** (*DMMs*).
- Diseño automático de **configuraciones de cache**.
- Creación automática de **máquinas de estados finitas** (*FSMs*).
- Construcción de reglas en **sistemas de trading**.
- Obtención de modelos en **entornos dinámicos**.
- Problemas de regresión simbólica.
- Algoritmos de comportamiento en robots.
- Evolución de reglas para reconocimiento de sonidos y palabras.
- Evolución de curvas fractales de alta dimensionalidad.

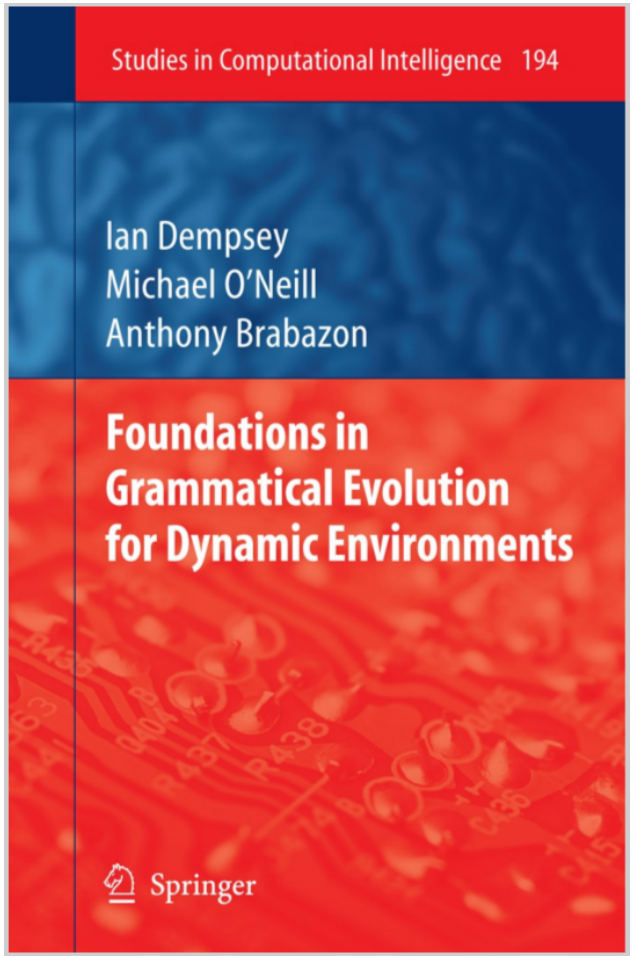
```
<codigo>      ::=  <codigo><indicador> | <indicador>
<indicator> ::=  <MAC> | <MACD> | <RSId> | <RSIo>| <SR> | <VP>

<MAC>   ::=   "MAC("<Param1>","<Param2>","<Peso>");"
<MACD> ::=   "MACD("<Param3>","<Param4>","<Param5>","<Peso>");"
<VP>   ::=  "VP("<Param6>","<Peso>");"
<SR>   ::=   "SR("<Param7>","<Param8>","<Param9>");"
<RSIo> ::=   "RSI(false,"<Param10>","<Param11>","<Param12>","<Peso>");"
<RSId> ::=   "RSI(true," <Param13> ","<Param14>","<Peso> ");"
```

Aplicaciones

- Obtención de **modelos para glucosa** en diabéticos.
- Diseño automático de **gestores de memoria dinámica** (*DMMs*).
- Diseño automático de **configuraciones de cache**.
- Creación automática de **máquinas de estados finitas** (*FSMs*).
- Construcción de reglas en **sistemas de trading**.

- Obtención de modelos en **entornos dinámicos**.
- Problemas de regresión simbólica.
- Algoritmos de comportamiento en robots.
- Evolución de reglas para reconocimiento de sonidos y palabras.
- Evolución de curvas fractales de alta dimensionalidad.



Preguntas

Fundamentos de la Evolución Gramatical

José Manuel Colmenar

josemanuel.colmenar@urjc.es

Universidad
Rey Juan Carlos

Adaptive and Bioinspired Systems Research Group