# Programare evolutivă și programare genetică

Motto:

"How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?"

— Attributed to Arthur Samuel (1959)

# Programare evolutivă

Istoric:

L. Fogel (1960) – dezvoltă metode de generare a unor sisteme cu comportament inteligent pornind de la ideea evoluției naturale (precursor al programării genetice)

D. Fogel (1990) – programarea evolutivă este orientată mai mult înspre rezolvarea unei clase mai largi de probleme (în particular optimizare numerică – similar cu strategiile evolutive)

Specific

- Codificarea este specifică problemei de rezolvat
- Folosesc doar mutație (întrucât modelează evoluția la nivelul speciilor și nu la nivelul indivizilor cum se întâmplă în cazul algoritmilor genetici si ai strategiilor evolutive)

# Programare evolutivă

Prima direcție:

- Tehnica de generare automată a unui sistem cu comportament inteligent
- Comportament inteligent: abilitatea de a se adapta la mediu
- Adaptabilitatea la mediu presupune capacitate de predicție
- Fitness-ul este o măsură asociată comportamentului sistemului

Exemplu: dezvoltarea unor sisteme de tipul automatelor finit deterministe (AFD)

AFD = (S, I, O, T,s0)

S – mulțimea stărilor

I – alfabet de intrare

O – alfabet de ieșire

T:SxI->SxO - reguli de tranziție

s0 – stare inițială

# Programare evolutivă

Exemplu:

Problema parității: AFD care verifică dacă un șir de biți conține un număr par de biți egali cu 1

- S={par,impar}
- I={0,1}
- O={0,1}

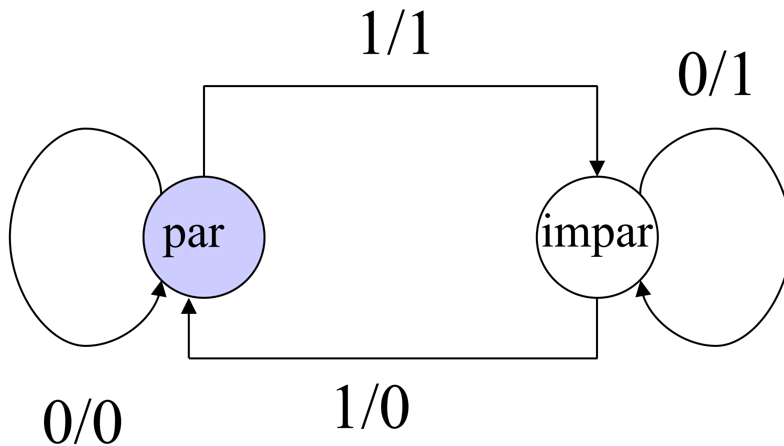
Interpretare răspuns:

stare finală = 0 (șirul are un număr par de elemente egale cu 1)

stare finală = 1 (șirul are un număr impar de elemente egale cu 1)

# Programare evolutiva

Diagrama de stări = multigraf etichetat




Proiectarea algoritmului:

- Se aleg: S, I, O

Inițializarea populatiei: se generează aleator AFD-uri:

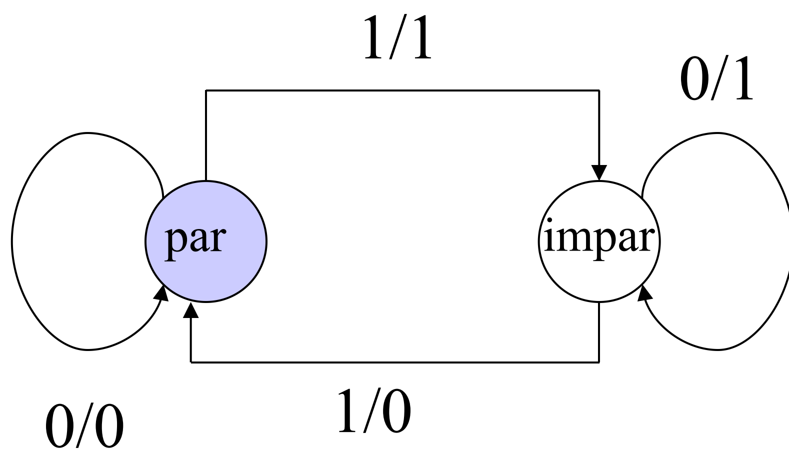
- Alegere etichete noduri
- Conectare noduri
- Etichetare arce

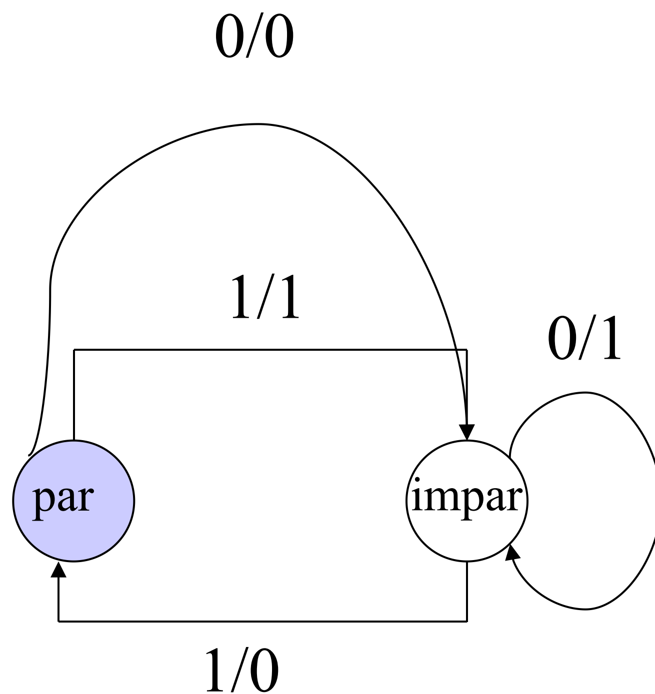
Mutație:

- Modificare simbol ieșire
- Adăugare/eliminare stare
- Modificare tranziție (schimbarea stării destinație)
- Modificarea stării inițiale

# Programare evolutivă

Exemplu mutație:



# Programare evolutivă

Specific codificare: elementele populatiei sunt grafuri etichetate

Reprezentare: explicită sau implicită

- Număr fixat de noduri + număr fixat de muchii => se modifică doar etichetele muchiilor:
  - Vector de dimensiune fixată conținând etichetele muchiilor (de regulă sunt ponderi ce conțin valori reale) = liniarizarea matricii de ponderi
- Număr fixat de noduri + număr variabil de muchii
  - Vector de dimensiune fixată (egală cu numărul maxim de muchii) ce conține etichetele muchiilor (mulțimea etichetelor conține o valoare specifică pentru muchii eliminate)
- Număr variabil de noduri => se modifică atât indicatorii de prezență ai nodurilor cât și etichetele muchiilor
  - Vector binar de dimensiune egală cu nr maxim de noduri (indicatori de prezență a nodurilor)
  - Vector ce conține etichetele asociate muchiilor

# Programare evolutivă

Mutație:

- Modificarea ponderilor muchiilor (prin inlocuire aleatoare in cazul valorilor discrete sau perturbare aditiva in cazul valorilor reale)
- Complementarea unor valori din vectorul cu indicatori de prezență a nodurilor => adăugare/ eliminare noduri

Observație: fiecare dintre variantele de mutație are o anumită probabilitate de aplicare. De exemplu:

- p1: modificare etichetă muchie
- p2: adăugare muchie
- p3: eliminare muchie
- p4: adaugare nod
- p5: eliminare nod

(p1+p2+p3+p4+p5=1)

# Programare evolutivă

Incrucisare:

este mai dificil de implementat în cazul grafurilor decât în cazul structurilor liniare

- in general nu se foloseste la programarea evolutivă, dar dacă se decide utilizarea acestui operator, atunci trebuie ținut cont de:

- In cazul grafurilor cu structură fixată (pe baza matricii de adiacență) => reprezentare liniară => se pot aplica operatorii clasici de încrucișare/recombinare
- In cazul grafurilor care nu sunt reprezentate prin structuri statice, ci sunt reprezentate de exemplu prin liste de adiacență: se pot defini operatori care realizează interschimbarea unor subgrafuri (se interschimba subsetul de muchii asociat aceluiasi set de noduri)

# Programare evolutivă

Evaluarea unei configurații:

- simularea comportării pe un set de testare
- calitatea configurației este direct proporțională cu rata de succes

Starea actuală: această variantă de programare evolutivă a fost redirecționată înspre aplicații care vizează proiectarea evolutivă a unor structuri de calcul (ex: rețele neuronale)

# Programare evolutivă

A doua direcție: metode similare strategiilor evolutive caracterizate prin:

- se utilizează doar mutație (fără recombinare)
  - mutația se bazează pe perturbație aleatoare (x’=x+N(0,s))
  - s depinde invers proporțional de valoarea fitness-ului

- pornind de la o populație cu m elemente se generează prin mutație m urmași iar cele mai bune m elemente vor forma generația viitoare (sau selecție de tip turneu) – similar cu strategie (m+m)

- Variantele de tip MetaEP includ autoadaptarea parametrilor (similar cu autoadaptarea în strategiile evolutive)

# Programare evolutivă

MetaEP

$$(x_1,\ldots,x_n,s_1,\ldots,s_n) \rightarrow (x'_1,\ldots,x'_n,s'_1,\ldots,s'_n)$$

$$s'_i = s_i(1+\alpha N(0.1)),\ \alpha \cong 0.2$$

$$x'_i = x_i + s'_i\, N(0.1)$$

Obs: ulterior această variantă de mutație a fost înlocuită cu una bazată pe repartiția log-normală (similar auto-adaptării de la strategiile evolutive)

# Programare genetică

Istoric: J. Koza (1990)

Informații: www.genetic-programming.org,
http://www.cs.bham.ac.uk/~wbl/biblio/

Scop: proiectarea în manieră evolutivă a unor metode de calcul (ex: expresii, programe, algoritmi) sau structuri (ex: circuite, arbori de decizie)

Specific: elementele populației sunt de cele mai multe ori structuri arborescente

Exemplu problemă: regresie simbolică

# Programare genetică

Regresie numerică

Date de intrare:

- perechi de valori: (arg, val)
- model de dependență (ex: liniară, pătratică etc)

Ieșire: valori ale parametrilor specifici modelului

Regresie simbolică

Date de intrare:

- perechi de valori: (arg, val)
- alfabet cu terminale (variabile, constante) și neterminale (operatori, funcții)

Iesire: expresie care descrie dependența dintre argumente și valori

# Programare genetică

## Regresie numerică

Date de intrare:

(1,3),(2,5),(3,7),(4,9)

Model: f(x)=ax+b

Răspuns: a=2 b=1

Căutare în spațiul valorilor parametrilor

## Regresie simbolică

Date de intrare:

(1,3),(2,5),(3,7),(4,9)

Alfabet: +,*,-,/,constante,x

Răspuns: 2*x+1

Căutare în spatiul formulelor ce satisfac anumite proprietăți

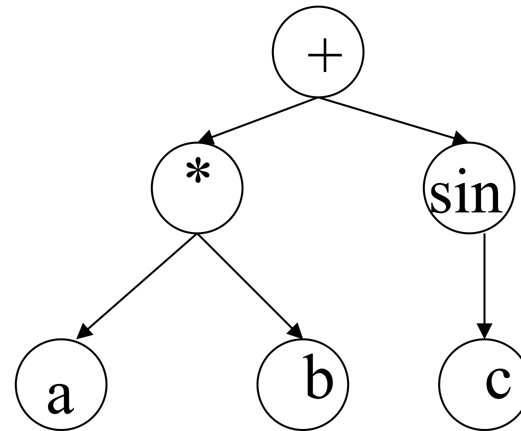
http://alphard.ethz.ch/gerber/approx/default.html

# Programare genetică

Reprezentare: elementele populației sunt de regulă structuri arborescente

Exemplu: expresie aritmetică

a*b+sin(c)




Elementele constitutive:

Neterminale: operatori și funcții

Terminale: variabile, constante (prestabilite sau generate aleator), funcții fără argument

Forma prefixată: +*a b sin c (parcurgere preordine)

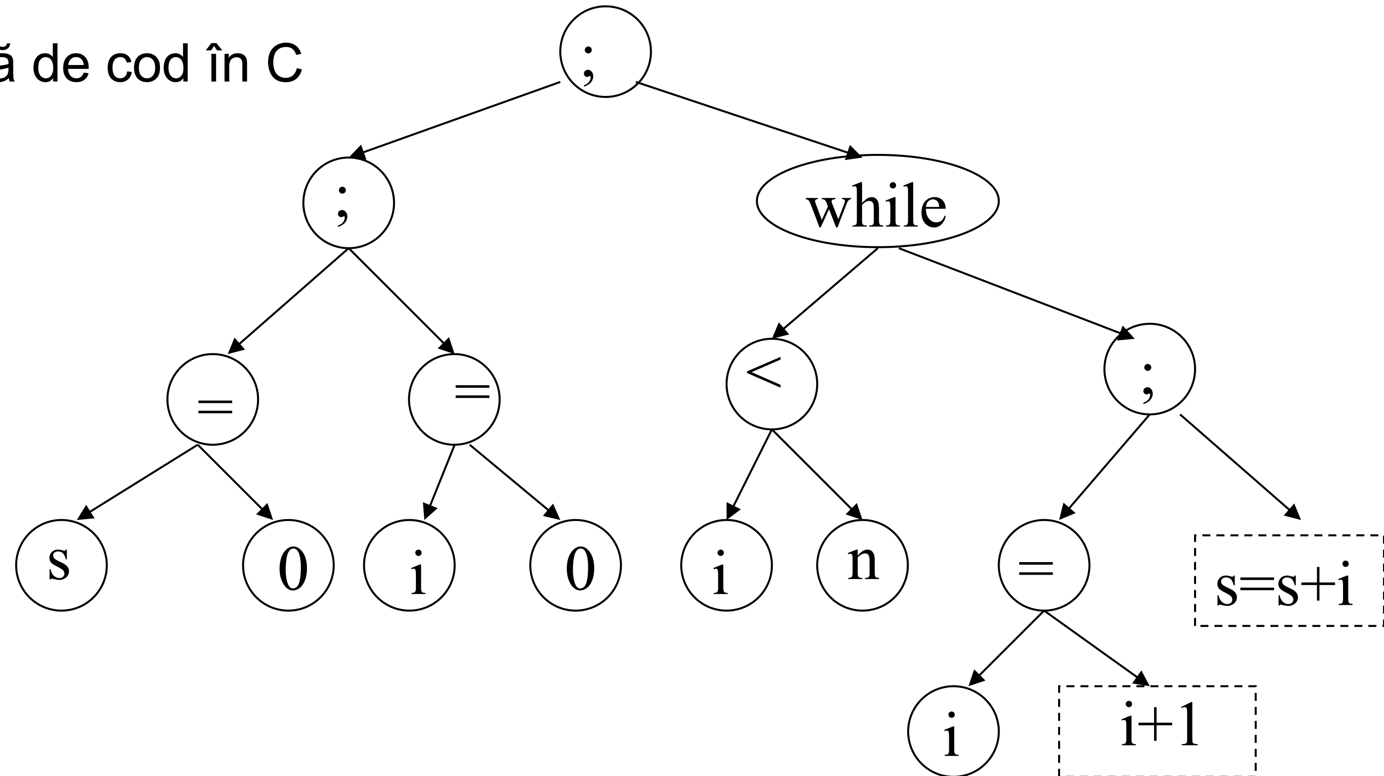
Forma postfixată: a b * c sin + (parcurgere postordine)

# Programare genetică

Reprezentare: elementele populației sunt structuri arborescente

Exemplu: secvență de cod în C

```
s=0;
i=0;
While (i<n)
   { i=i+1;
     s=s+i;
   }
```




Problema: structura arborescentă poate fi complexă chiar pentru programe simple

# Programare genetică

Sumar: set neterminale, set terminale

se aleg in funcție de problema de rezolvat

| Function Set | |
|---|---|
| *Kind of Primitive* | *Example(s)* |
| Arithmetic | +, *, / |
| Mathematical | sin, cos, exp |
| Boolean | AND, OR, NOT |
| Conditional | IF-THEN-ELSE |
| Looping | FOR, REPEAT |
| ⋮ | ⋮ |

| Terminal Set | |
|---|---|
| *Kind of Primitive* | *Example(s)* |
| Variables | x, y |
| Constant values | 3, 0.45 |
| 0-arity functions | rand, go_left |

# Programare genetică

Implementare:

- varianta clasică: LISP
- similar reprezentării prefixate a expresiilor

Dificultate: la inițializarea populației și la aplicarea operatorilor trebuie asigurată corectitudinea sintactică

T: terminale

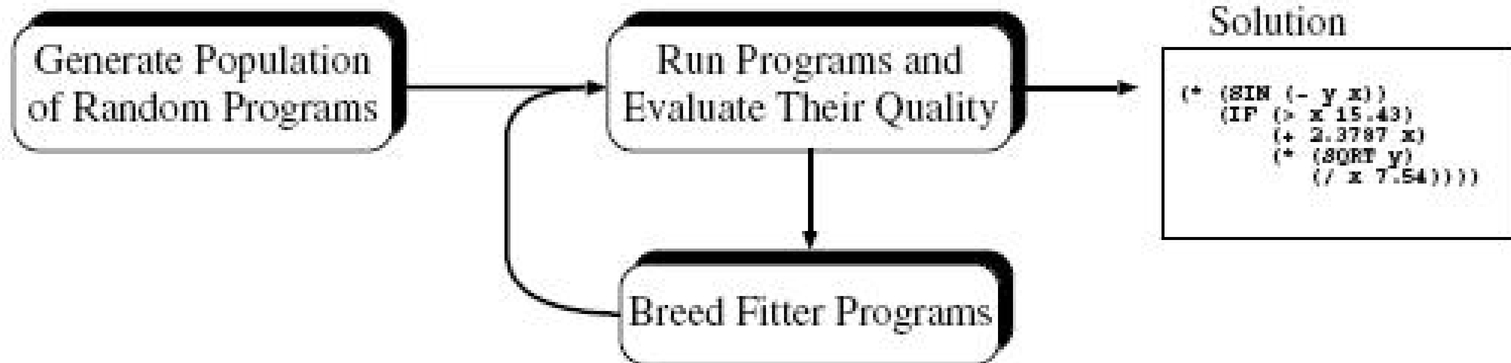
N: neterminale

A: adâncime arbore

Algoritm de generare a unui element

```
Generare(T,N,A)
IF A=0 THEN expr:=aleg(T)
ELSE
  fct:=aleg(N)
  IF (unar(fct)) THEN
          arg:=generare(T,N,A-1)
          expr:=(fct,arg)
  IF (binar(fct)) THEN
          arg1:=generare(T,N,A-1)
          arg2:=generare(T,N,A-1)
          expr:=(fct,arg1,arg2)
RETURN expr
```

# Programare genetică



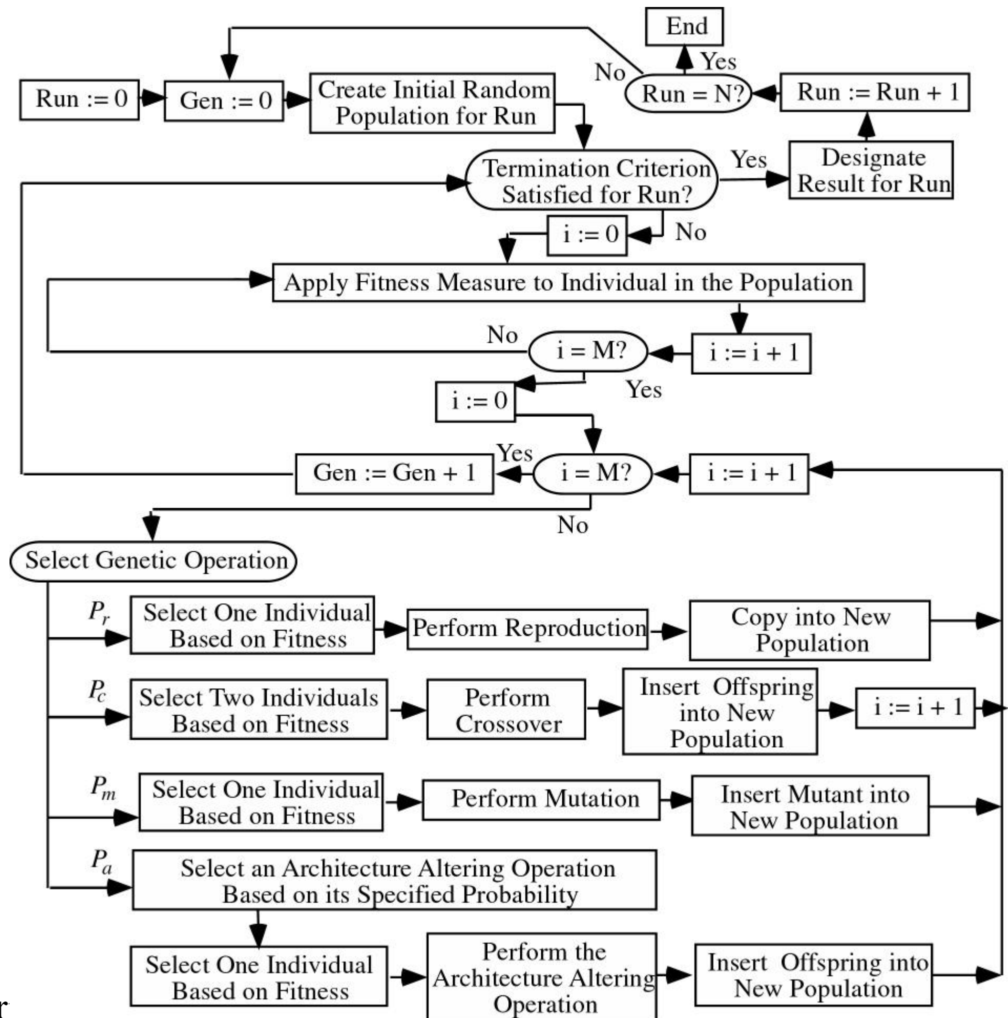
# Programare genetică

Structura generală unui algoritm de tip “programare genetica” [Koza, 2003]

Particularități: operatorii genetici se aplică alternativ (nu succesiv ca la AG si SE). Fiecare operator e selectat cu o anumită probabilitate după care se selectează elementele asupra căruia se aplică.

Exemple de operatori:

- Copiere directă în noua populație
- Incrucișare
- Mutație



# Programare genetică

Alte tipuri de elemente ale populației:

- Arbori de decizie
- Reguli de tip If-then
- Rețele neuronale, rețele bayesiene
- Expresii logice
- Diagrame binare de decizie
- Gramatici
- Sisteme de clasificare

# Programare genetică

Alte modalități de reprezentare:

- Succesiune de linii de cod (Linear Genetic Programming)
- Structuri care modelează ADN-ul (Gene Expression Programming)

# Programare genetică

LGP - Linear Genetic Programming [Brameier, Banzhaf, 2003]

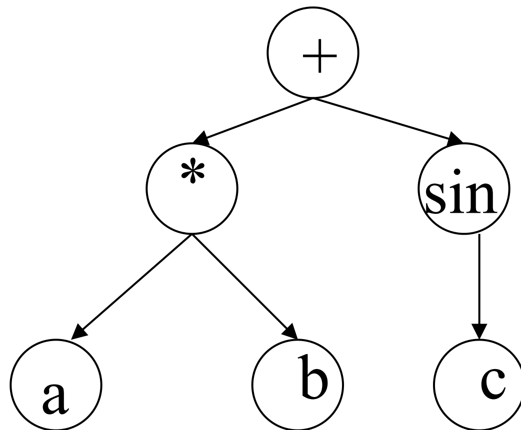
```
void gp(r)
  double r[8];
{
   ...
   r[0] = r[5] + 71;
// r[7] = r[0] - 59;
   if (r[1] > 0)
   if (r[5] > 2)
     r[4] = r[2] * r[1];
// r[2] = r[5] + r[4];
   r[6] = r[4] * 13;
   r[1] = r[3] / 2;
// if (r[0] > r[1])
//   r[3] = r[5] * r[5];
   r[7] = r[6] - 2;
// r[5] = r[7] + 15;
   if (r[1] <= r[6])
     r[0] = sin(r[7]);
}
```

Specific:

- Pentru generare de programe imperative (ex: C sau limbaje de asamblare)
- Instrucțiunile de atribuire modifică valorile aflate în variabile indexate corespunzătoare regiștrilor de memorie
- Alte instrucțiuni: decizionale (if) și de control (goto)
- Liniile comentate corespund prelucrărilor ce nu influențează rezultatul (similare intronilor – porțiuni necodante – din ADN)
- Incrucișare: variantă a încrucișării cu un punct de tăietură adaptată pt. cromozomi de dimensiuni diferite

# Programare genetica

GEP - Gene Expression Programming (C. Ferreira, 2001):




Cromozom:

- Constituit din mai multe “gene” de lungime fixată
- Fiecare genă conține o parte de antet (head) și o parte finală (tail)
- Antetul conține h elemente (atât neterminale cât și terminale); coada conține doar terminale (h*(n-1)+1, n=aritatea maxima a operatorilor din antet

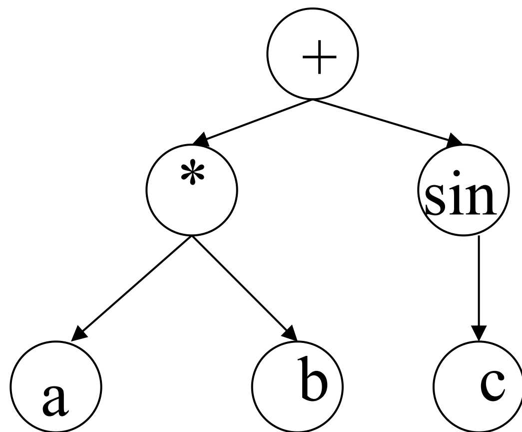
Exemplu: gena de lungime 13 = 6+(6*(2-1)+1)=h+(h*(n-1)+1)

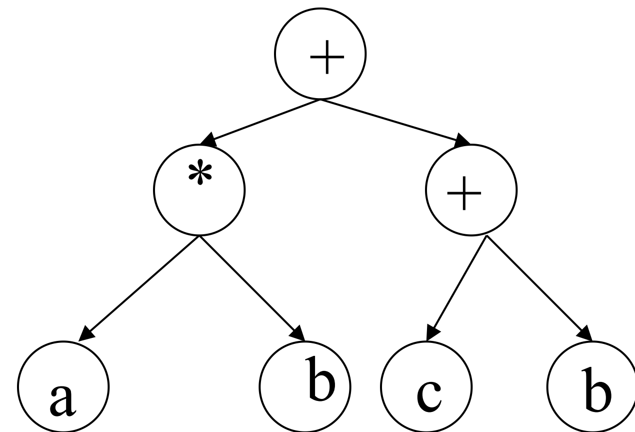
+ * sin a b c b a c c b a a

- Primele 6 elemente corespund cu structura (parcurgere în lățime a arborelui)
- Restul elementelor sunt terminale (neutilizate deocamdată în construirea fenotipului)

# Programare genetică

GEP: permite generarea ușoara de configurații corecte din punct de vedere sintactic prin extinderea antetului în zona finală




+ * sin a b c b a c c b a a

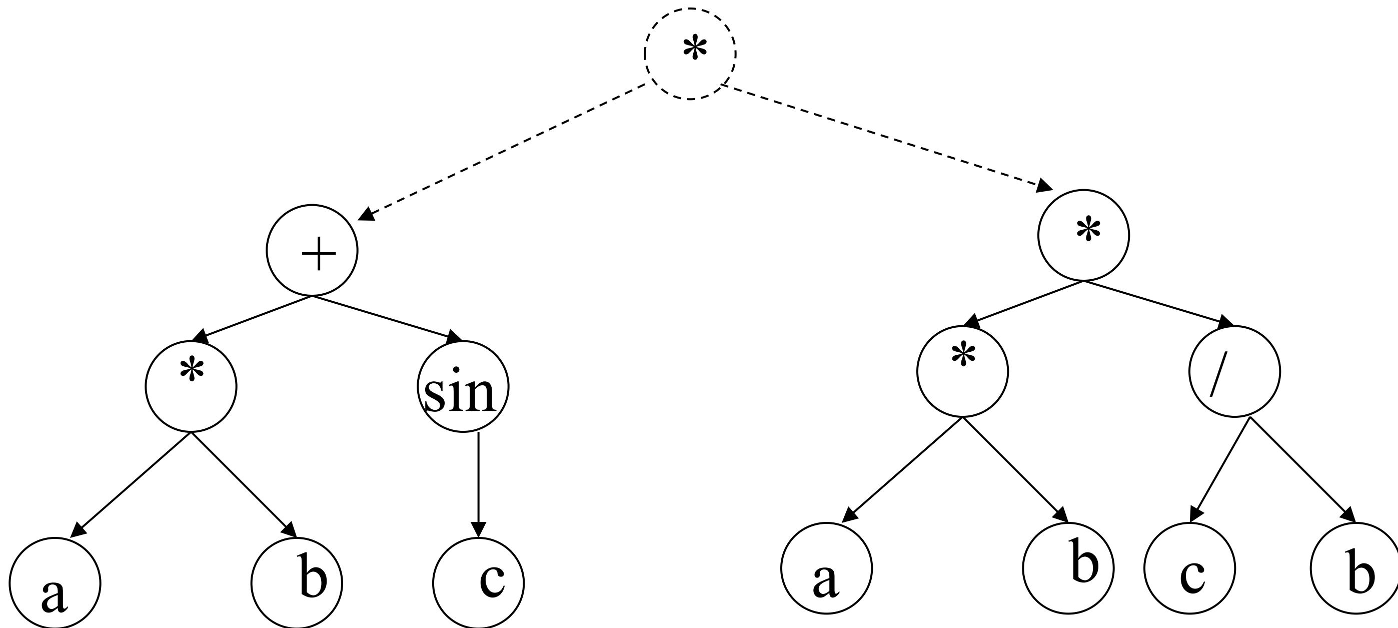
+ * + a b c b a c c b a a

# Programare genetică

GEP: cromozom constituit din două gene:

+ * sin a b c b a c c b a a * * / a b c b a c c b a a

Structura obținută prin combinarea structurilor definite de către cele două gene componente



# Programare genetică

Stabilirea calității unui element:

- evaluarea expresiei (simularea structurii) pentru un set de date de testare  (pentru care se cunosc atât argumentele cât și rezultatele)

- un element este cu atât mai bun cu cât valoarea obținută prin evaluarea expresiei/ programului asociat este mai apropiată de valoarea dorită

# Programare genetică

Evaluare expresie:

**Algorithm 3** Typical interpreter for GP

```
procedure: eval( expr )
 1: if expr is a list then
 2:    proc = expr(1) {Non-terminal: extract root}
 3:    if proc is a function then
 4:       value = proc( eval(expr(2)), eval(expr(3)), ... ) {Function: evaluate
          arguments}
 5:    else
 6:       value = proc( expr(2), expr(3), ...) {Macro: don't evaluate arguments}
 7: else
 8:    if expr is a variable or expr is a constant then
 9:       value = expr {Terminal variable or constant: just read the value}
10:    else
11:       value = expr() {Terminal 0-arity function: execute}
12: return value
```

# Programare genetică

Selecție părinți: selecție proporțională sau de tip turneu (ca la algoritmii genetici)

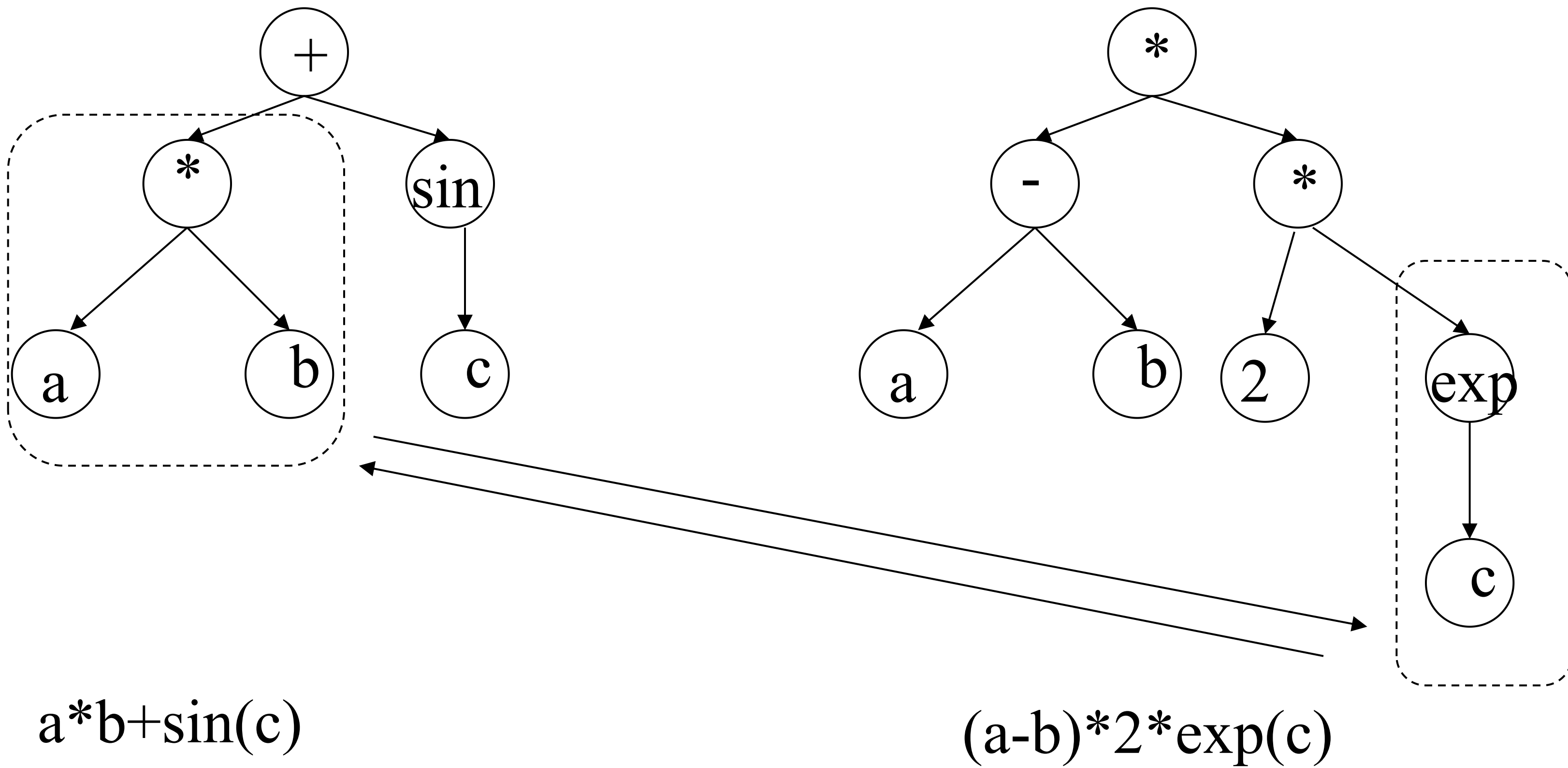
Selecție supraviețuitori: elementele create prin încrucișare și mutație sunt asimilate în noua populație

Variante:

- sincronă (generațională): se construiește o populație de urmași
- asincronă (steady state): pe măsură ce se construiește un nou element, el este asimilat în populație prin înlocuirea celui mai slab element din populația curentă (are caracter elitist)

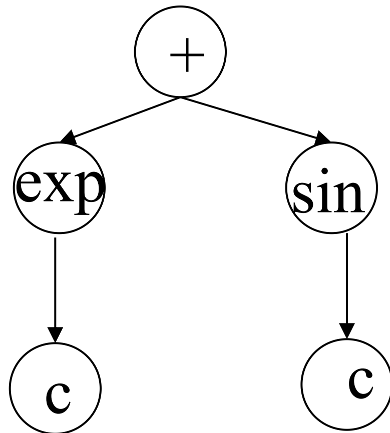
# Programare genetică

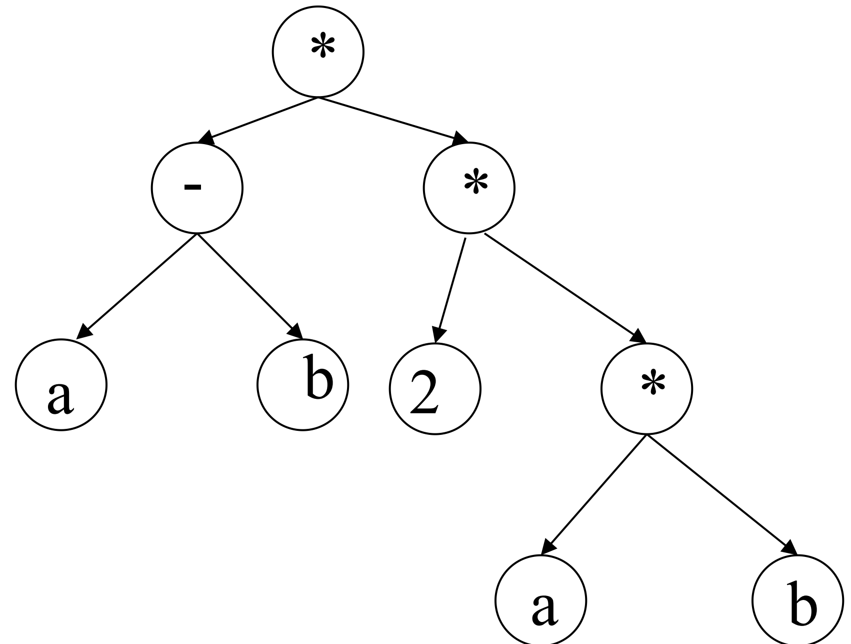
Recombinare: doi părinți (arbori) generează doi urmași (arbori) prin interschimbarea unor subarbori



# Programare genetică

Recombinare: doi părinți (arbori) generează doi urmați (arbori) prin interschimbarea unor subarbori




exp(c)+sin(c)

(a-b)*(2*(a*b))

# Programare genetică

Recombinare:

Variantele prefixate ale părinților și urmașilor

| | |
|---|---|
| + * a b sin c | * - a b * 2 exp c |
| + exp c sin c | * - a b * 2 * a b |

Obs. Este oarecum similară cu încrucișarea de la algoritmii genetici însă porțiunile care se interschimbă nu au aceeași dimensiune

# Programare genetică

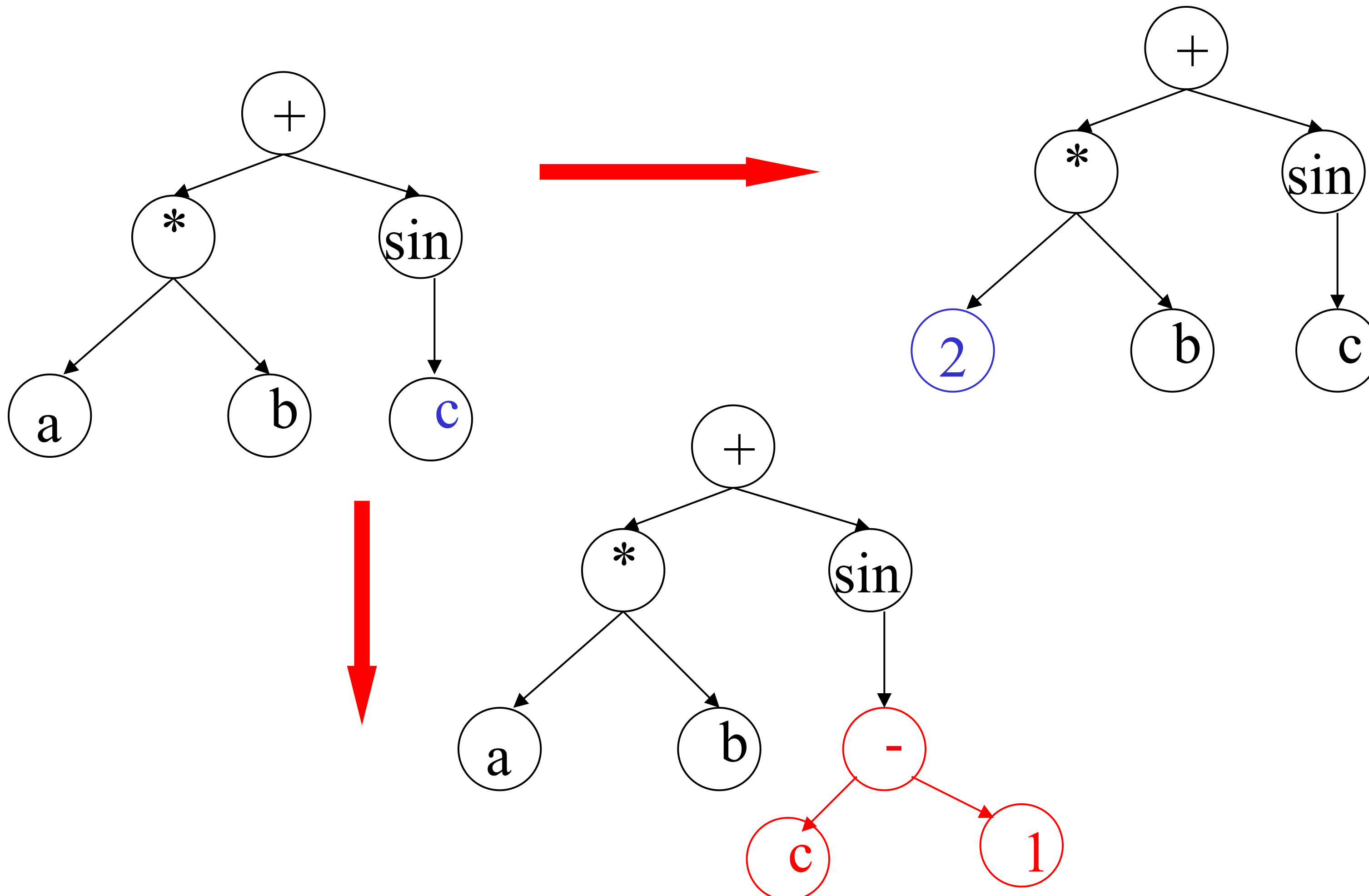
Mutație: constă în modificarea aleatoare a unor componente

- Inlocuirea etichetei unui nod frunză cu un alt simbol terminal (în cazul terminalelor de tip constantă această modificare poate fi de tipul unei mutații specifice strategiilor evolutive)
- Inlocuirea unui nod terminal cu un arbore (creștere)
- Inlocuirea etichetei unui nod neterminal cu un simbol neterminal din aceeași clasă (operator unar, binar, etc.)
- Inlocuirea unui subarbore cu un nod terminal (eliminarea unei subexpresii - reducere)

Obs: poate fi interpretată și implementată ca o recombinare cu o structură generată aleator

# Programare genetica

Mutatie: constă în modificarea aleatoare a unor componente



# Programare genetică

Observație: în programarea genetică încrucișarea și mutația nu se aplică în mod necesar succesiv asupra aceluiași element ci mai degrabă alternativ (elementele create prin încrucișare respectiv mutație intră în competiție pentru supraviețuire)

Problema: explozia dimensiunii structurilor (structurile mari au tendința să domine populația)

Soluții:

- se impune un prag privind complexitatea structurii (ex: adâncimea arborelui) și nu se acceptă configurații care depășesc pragul
- se introduce un factor de penalizare la evaluarea fitnessului care defavorizează structurile mari

# Programare genetică

Exemple aplicații:

- Extragere modele din date (reguli de decizie) cu aplicații în analiza financiară
- Proiectarea circuitelor analogice
- Sinteza sisteme robuste
- Evolvable hardware

# Programare genetică

Exemple aplicații:

- generare programe specifice calculului paralel (ex: retele de sortare)
- determinare reguli de tranziție pentru automate celulare
- generare strategii pentru sisteme multi-agent
- generare strategii de joc
- generare algoritmi cuantici
- generare filtre pentru prelucrarea semnalelor