Evolutionary Programming and Genetic Programming

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)

Evolutionary Programming

The origins:

L. Fogel (1960) – development of methods, inspired by the natural evolution, which generate automatically systems with some intelligent behavior;

D. Fogel (1990) – in the last years the evolutionary programming became more oriented toward solving problems (optimization and design)

Particularities

- Various encoding variants (e.g. real vectors, state diagrams, neural networks structures)
- Based only on mutation, no recombination
- Current variants: self-adaptive

Evolutionary Programming

First (traditional) direction :

- Evolve systems (e.g. finite state machine) with prediction abilities
- The fitness of such a structure is measured by analyzing the behavior of the system = prediction abilities
- The fitness is a quality measure related to the behaviour of the system

Finite State Machines (FSM):

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

S – set of states

I – input alphabet

O – output alphabet

T:SxI->SxO - transition rules

s0 – initial state

Evolutionary Programming

A simple test problem:

design a FSM to check if a binary string has an even or an odd numbers of elements equal to 1 (parity problem)

- S={even,odd}
- I={0,1}
- O={0,1}

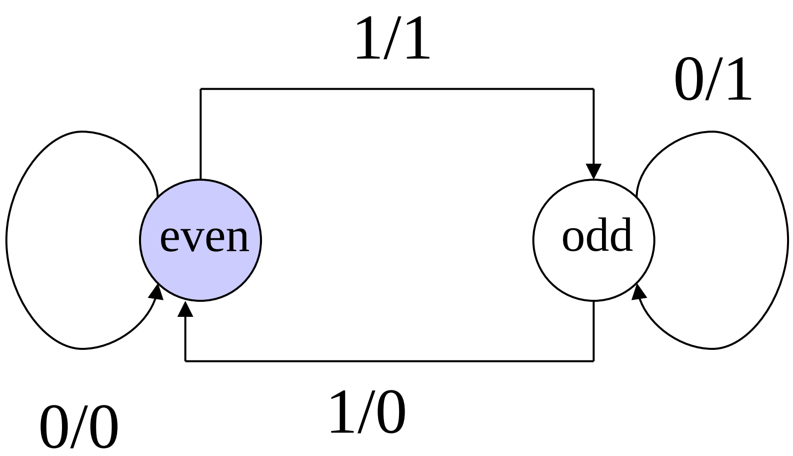
FSM output:

final state = 0 (the sequence has an even number of 1)

final state = 1 (the sequence has an odd number of 1)

Evolutionary Programming

State diagram = labeled directed graph



EP Design:

- choose: S, I, O

Population initialization: generate random FSMs

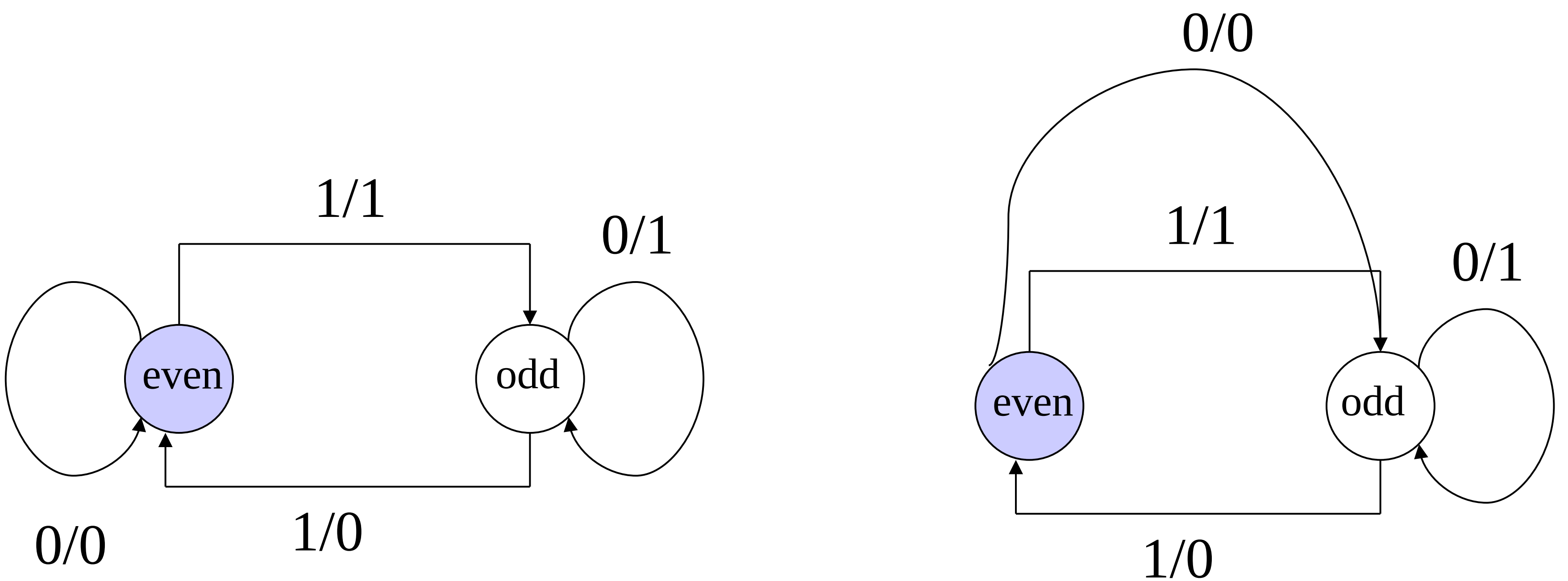
- Generate labels for nodes
- Generate arcs
- Generate labels

Mutation:

- Mutation of the output symbol
- Redirect an arc (mutate the target node)
- Add/eliminate nodes
- Change the initial state

Evolutionary Programming

Mutation example: change the target node of an arc



Evolutionary Programming

Evaluation of a configuration:

- simulation for a test set
- the fitness is considered to be proportional with the success rate

Current status in the field: this direction of EP is no more of actuality; it has been redirected to the evolutionary design of computational structures (e.g. neural networks)

Evolutionary Programming

Second (current) direction: it is related to optimization methods similar to evolution strategies

- there is only a mutation operator (no recombination)
  - the mutation is based on random perturbation of the current configuration ($x'=x+N(0,s)$)
  - s is inversely correlated with the fitness value (high fitness leads to small s, low fitness leads to large values for s)

- starting from a population with m elements, by mutation are constructed m children and the survivors are selected from the 2m elements by tournament or by truncation.

- There are self-adaptive variants, called MetaEP; these variants are similar to self-adaptive Evolution Strategies

Evolutionary Programming

MetaEP

$$(x_1,\ldots,x_n,s_1,\ldots,s_n)\rightarrow(x'_1,\ldots,x'_n,s'_1,\ldots,s'_n)$$
$$s'_i = s_i(1+\alpha N(0.1)),\ \ \alpha \cong 0.2$$
$$x'_i = x_i + s'_i\, N(0.1)$$

Remark: currently the normal mutation used to self-adapt the control parameters has been replaced with a log-normal distribution (as in the case of SE)

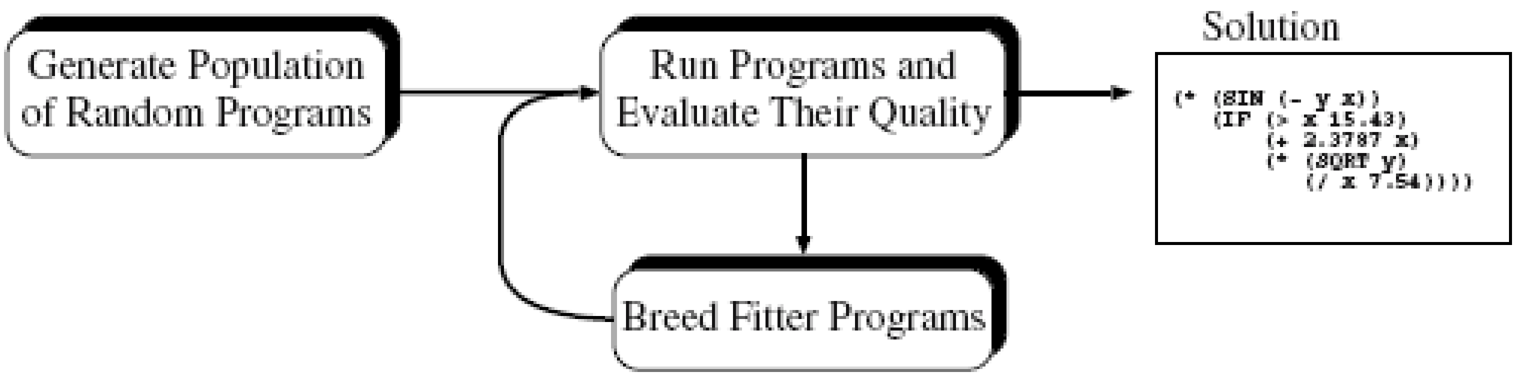
Genetic Programming

Principal contributor: J. Koza (1990)

Official web site: www.genetic-programming.org

- GP is an automated method for creating a working computer program from a high-level problem statement of a problem.
- GP starts from a high-level statement of “what needs to be done” and automatically creates a computer program to solve the problem.

Genetic Programming



The result is a program or an "executable" expression

Genetic Programming

Numeric regression

Input data:
- pairs of values: (arg, val)
- model which depends on some parameters(e.g.: linear model, quadratic model etc)

Output: values of the model parameters

Symbolic regression

Input data:
- pairs of values : (arg, val)
- alphabets of terminals (variables, constants) and nonterminals (operators, functions)

Output: expression which describes the dependence between output variable (predicted value) and the input variable (predictor)

Genetic Programming

Numerical regression

Input data:
(1,3),(2,5),(3,7),(4,9)

Model: f(x)=ax+b

Result: a=2 b=1

Search in the parameter space

Symbolic regression

Input data:
(1,3),(2,5),(3,7),(4,9)

Alphabet: +,*,-,/,constants,x

Result: 2*x+1

Search in the space of expressions

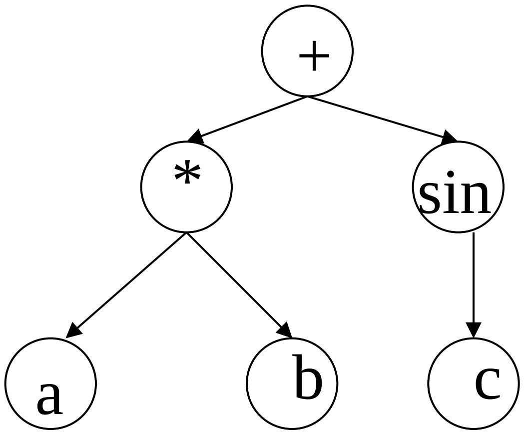
http://alphard.ethz.ch/gerber/approx/default.html

Genetic Programming

Encoding: the individuals are usually tree-like structures

Example 1: arithmetical expression

a*b+sin(c)



Components:

Nonterminals: operators and functions

Terminals: variables, constants (fixed or randomly generated), 0-arity functions

Prefixed form: +*a b sin c (preorder)

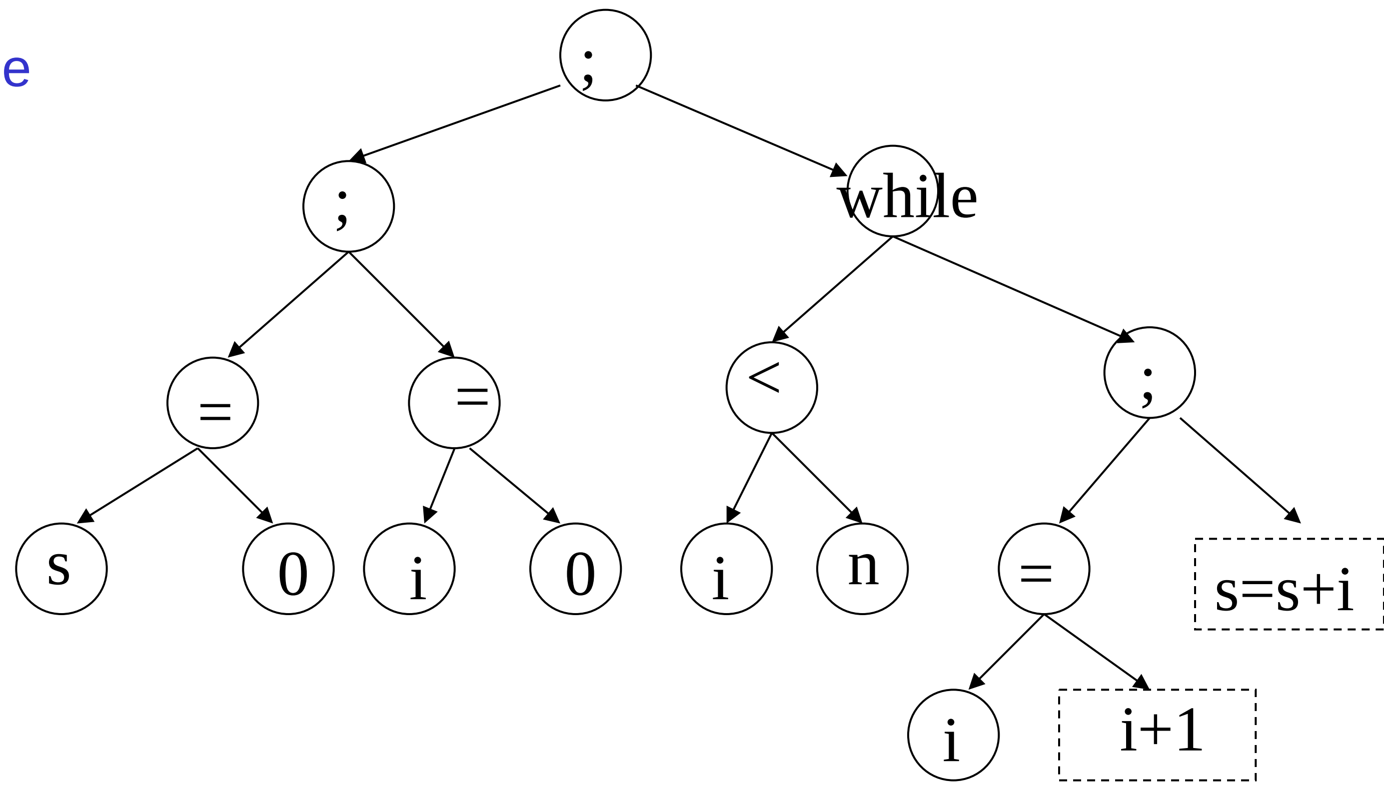
Postfixed form: a b * c sin + (postorder)

Genetic Programming

Encoding: the individuals are usually tree-like structures

Example 2: C code

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



Problem: the tree representation can be complex even for simple programs

Genetic Programming

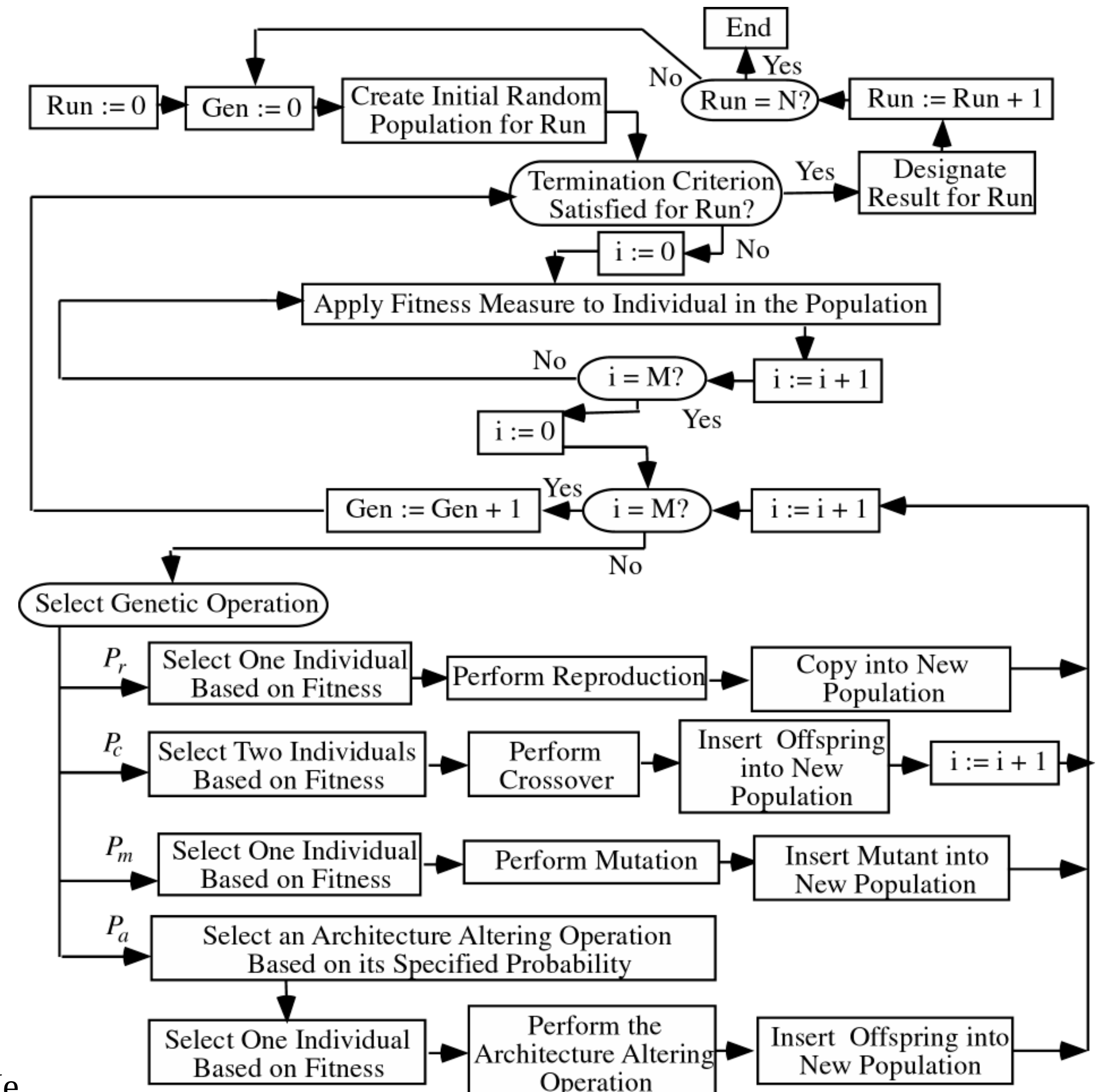
Summary: the terminals and nonterminals sets are chosen depending on the problem to be solved

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

Genetic Programming

Overall structure of a GP algorithm [Koza, 2003]



Genetic Programming

Implementation:

- classical variant: LISP
- lists corresponding to prefixed description of expressions

Difficulty: all elements should be syntactically correct

Generation function - parameters

T: terminals

N: nonterminals

A: tree depth

```
Generate(T,N,A)
IF A=0 THEN expr:=choose(T)
ELSE
  fct:=choose(N)
  IF (unary(fct)) THEN
           arg:=generate(T,N,A-1)
           expr:=(fct,arg)
  IF (binary(fct)) THEN
          arg1:=generate(T,N,A-1)
          arg2:=generate(T,N,A-1)
          expr:=(fct,arg1,arg2)
RETURN expr
```

Genetic Programming

Other types of population elements:

- Decision trees
- If-then rules
- Neural networks
- Logical expressions
- Binary decision diagrams
- Grammars

Genetic Programming

Fitness computation:

- the expression (phenotype) corresponding to each chromosome (genotype) is evaluated for a test data set

- the fitness of a chromosome is higher if the value obtained by evaluating the expression is close to the desired value

Genetic Programming

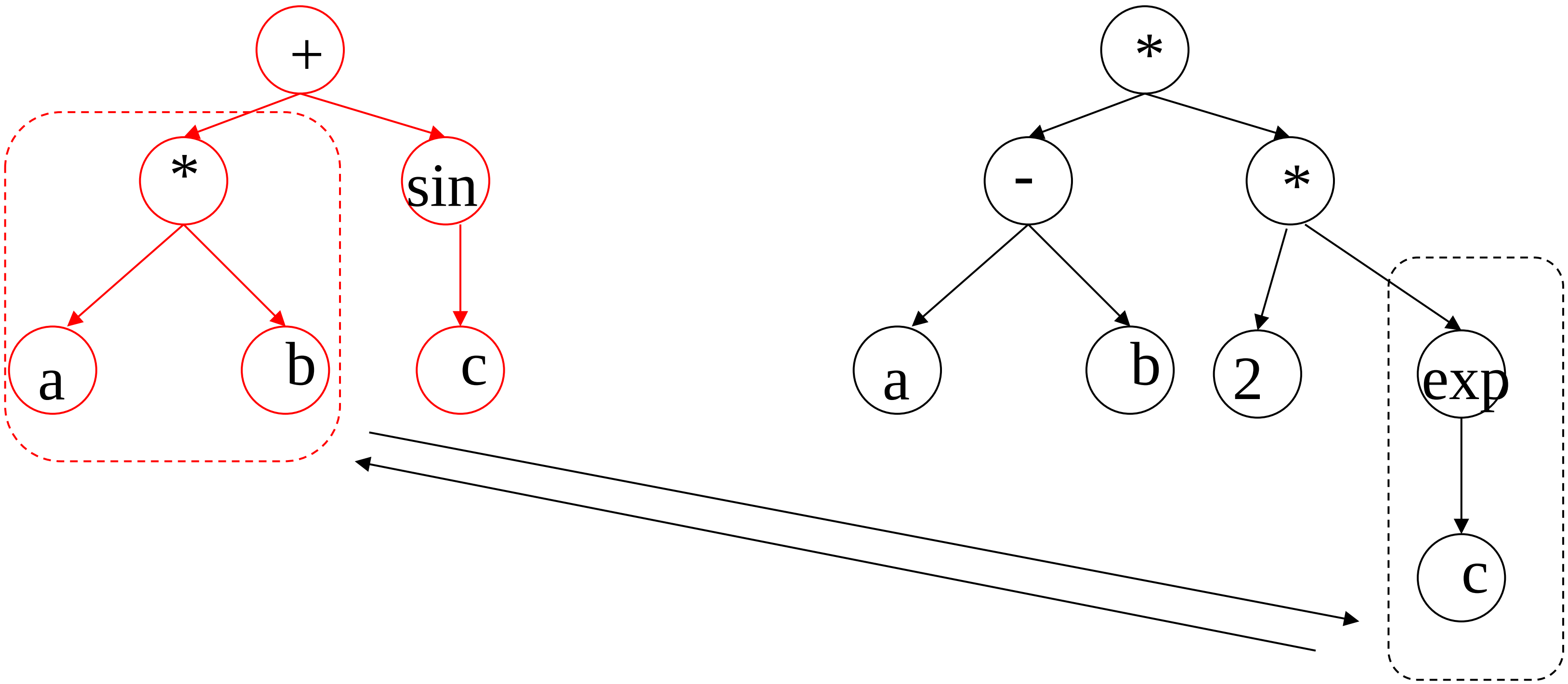
Evaluation:

Algorithm 3 Typical interpreter for GP

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

Genetic Programming

Crossover: two parents (trees) generate two offspring (also trees) by swapping some subtrees

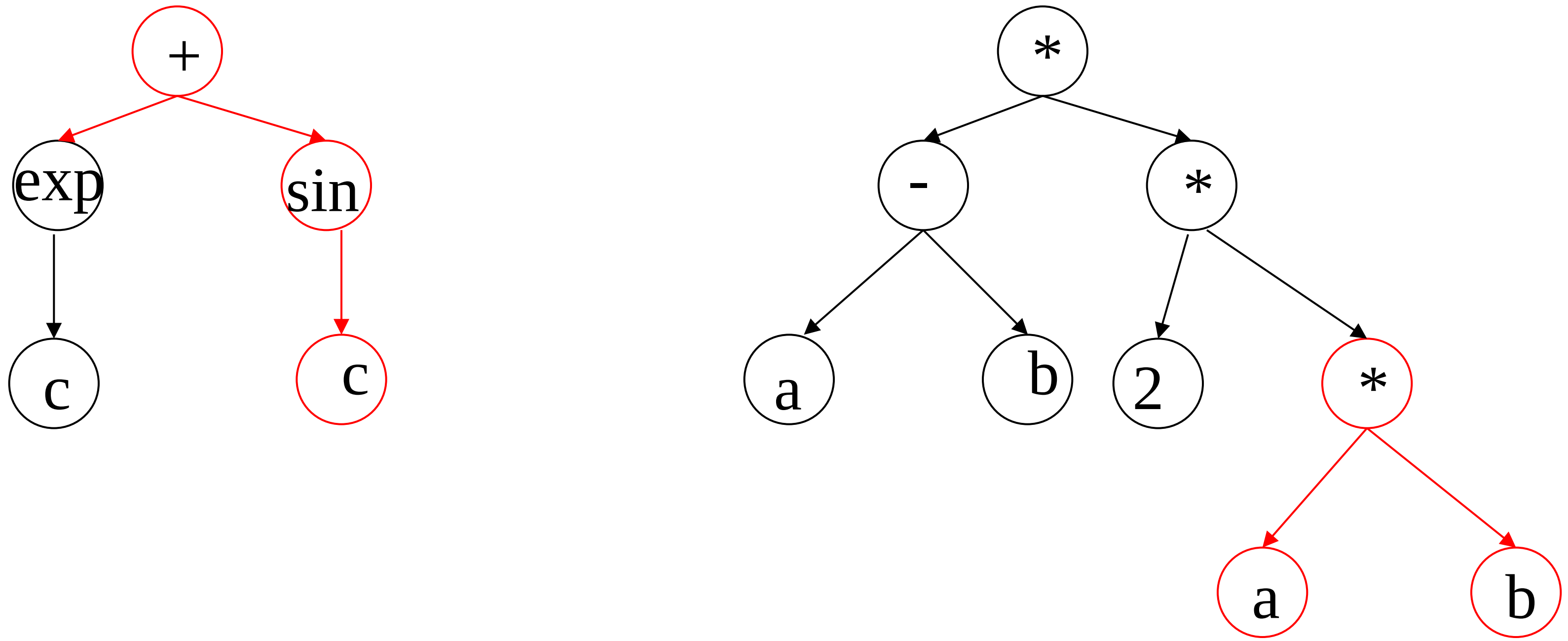


a*b+sin(c)

(a-b)*2*exp(c)

Genetic Programming

Crossover: two parents (trees) generate two offspring (also trees) by swapping some subtrees



exp(c)+sin(c)

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

Genetic Programming

Crossover:

Prefixed forms of parents and children

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

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

Remark. It is similar to the crossover used at GAs but the size for exchanged portions are usually different.

Genetic Programming

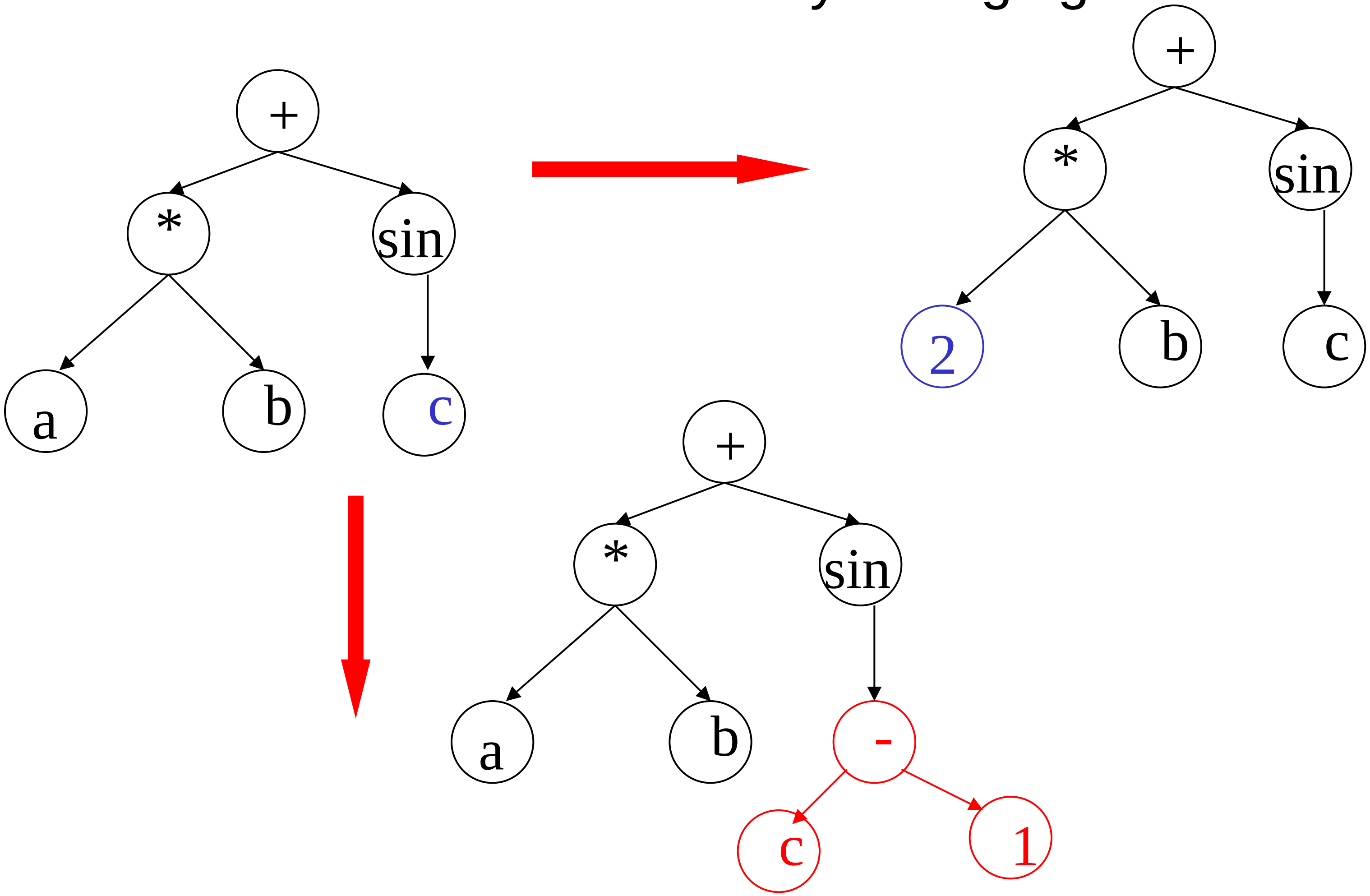
Mutation: consists of randomly changing some elements

- Change the symbol of a leaf node with another terminal symbol (in the case of constants this mutation could be as in the case of evolution strategies)
- Replace a leaf node with a tree (growing mutation)
- Replace the symbol corresponding to an internal node with another nonterminal from the same class (function with the same arity)
- Replace a subtree with a terminal node (pruning mutation)

Remark: the mutation can be implemented by a crossover with a randomly generated element

Genetic Programming

Mutation: consists of randomly changing some elements



Genetic Programming

Bloat problem: the complex structures become dominant in the population

Solutions:

- Use a threshold for the structure complexity (e.g. tree depth) and reject all structures larger (deeper) than the threshold
- Use a penalty term depending on the structure complexity in the fitness computation; this term will penalize the complex structures

Genetic Programming

GP related approaches:

- Linear Genetic Programming
- Gene Expression Programming [http://www.gene-expression-programming.com/]
- Cartesian Genetic Programmming [http://www.cartesiangp.co.uk/]
- Multi-expression Programming [http://www.mep.cs.ubbcluj.ro/]
- Grammatical Evolution [http://www.grammatical-evolution.org/]

Genetic Programming

Linear Genetic Programming [Brameier, Banzhaf, 2003]

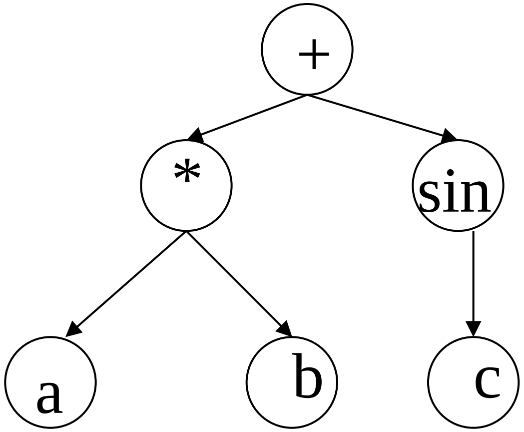
```
void gp(r)
  double r[8];
{
   ...
   r[0] = r[5] + 71;
// r[7] = r[0] - 59;
   if (r[1] > 0)
   if (r[5] > 2)
     r[4] = r[2] * r[1];
// r[2] = r[5] + r[4];
   r[6] = r[4] * 13;
   r[1] = r[3] / 2;
// if (r[0] > r[1])
//   r[3] = r[5] * r[5];
   r[7] = r[6] - 2;
// r[5] = r[7] + 15;
   if (r[1]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

Particularities:

- Used to generate programs as sequences of lines (e.g. like in assembling languages)
- The operations involves registers
- Instructions: if and goto
- The commented lines correspond to processing steps which do not influence the final result (similar to noncoding portions of DNA – the so-called introns)
- Crossover: uses a variant of single point crossover adapted for chromosomes with different lengths (the program is a chromosome, each line is a gene)

Genetic Programming

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



Chromosome:

- Consists of several genes of fixed length
- Each gene has a head and a tail
- The head contains h symbols (both terminals and nonterminals); the tail contains only terminals; the number of elements in the tail is h*(n-1)+1, n=the maximal arity of functions/operators which appears in the head

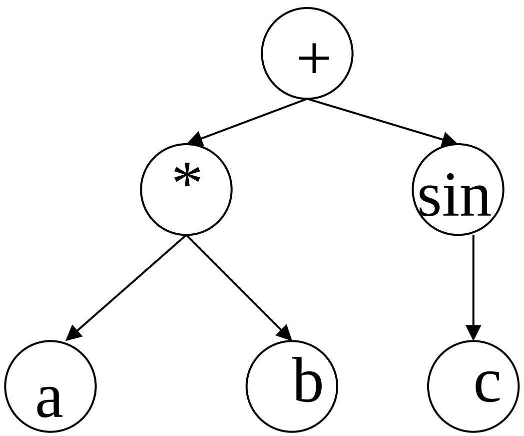
Example: gene of length 13 = 6+(6*(2-1)+1)=h+(h*(n-1)+1)

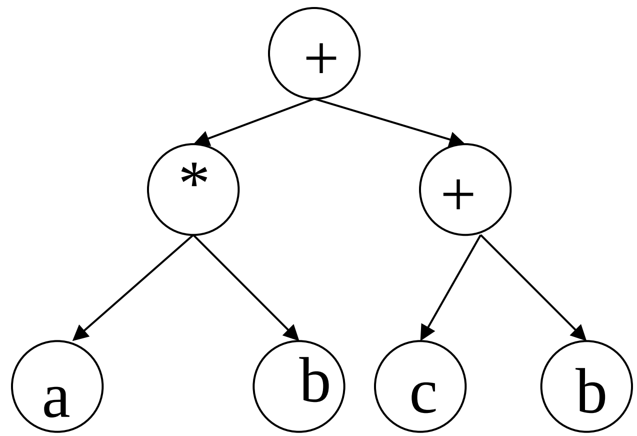
+ * sin a b c b a c c b a a

- The first 6 elements correspond with the expression (breadth first search of the tree)
- All other elements are terminal (unused in the genotype-phenotype conversion)

Genetic Programming

GEP: allow to generate syntactically correct expressions by extending the head over the symbols in the tail





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

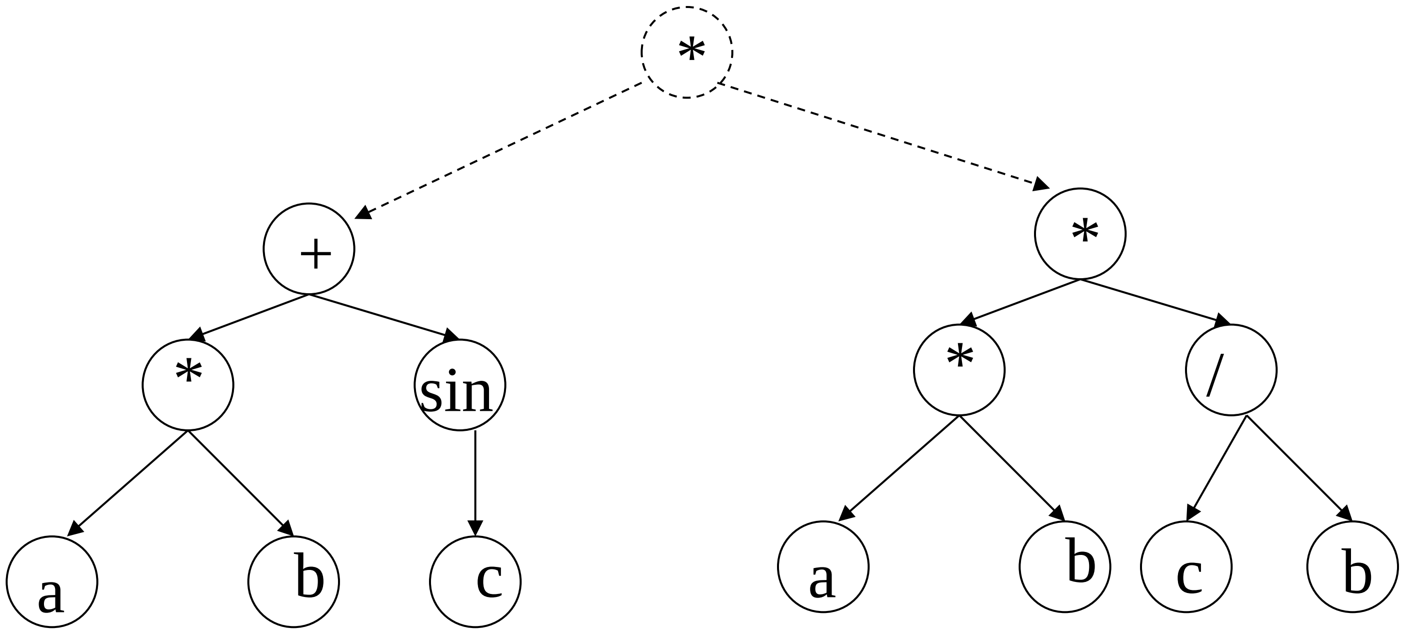
+ * + a b c b a c c b a a

Genetic Programming

GEP: chromosome consisting of two genes:

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

The phenotype corresponding to the chromosome is obtained by combining the genes corresponding to the two genes



Genetic Programming

Applications:

- Extracting models from data (e.g. predictive models)
- Extracting rules from data
- Electrical circuits design
- Robust systems synthesis
- Evolvable hardware

Genetic Programming

- parallel applications design
- cellular automata design
- signal/image processing filters design
- generation of multi-agent strategies
- generation of game strategies
- generation of quantum algorithms

Genetic Programming

Genetic Programming Software:

- Java: ECJ, TinyGP,
- Matlab: GPLab, GPTips
- C/C++: MicroGP
- Python: DEAP, PyEvolve