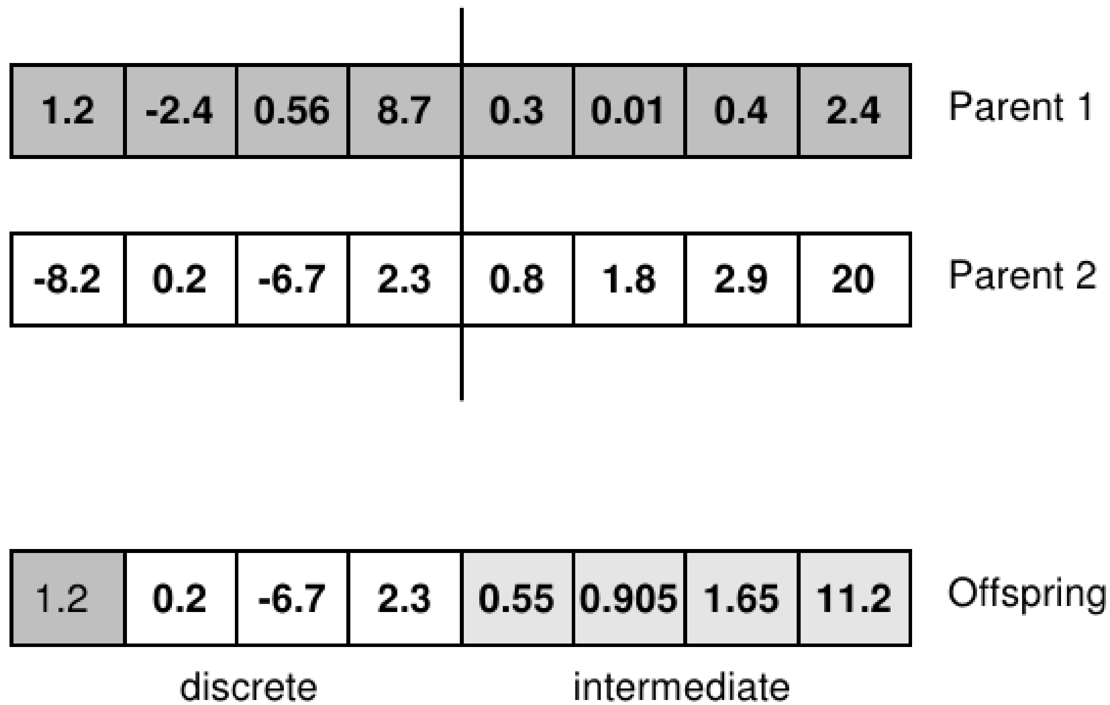
# ES Crossover / Recombination

- Application of operator creates **one** child (not two)
- Is applied $\lambda$ times to create an offspring population of $\lambda$ size (on which then mutation and selection is applied)
- Per offspring gene two parent genes are involved
- Choices:
  - combination of two parent genes:
    - average value of parents (*intermediate recombination*)
    - value of one randomly selected parent (*discrete recombination*)
  - choice of parents:
    - a different pair of parents for each gene (*global recombination*)
    - the same pair of parents for all genes

# ES Crossover / Recombination

- Default choice: discrete recombination on phenotype, intermediate recombination on strategy parameters



# GAs vs. ES

Genetic algorithms

- Crossover is the main operator
- Uses also mutation
- Encoding for problem representation
- Biased selection of the parents
- Algorithm parameters often fixed
- Selection → Crossover → Mutation

Evolution strategies

- Mutation is the main operator
- Uses also crossover (recombination)
- No encoding needed for problem representation
- Random selection of the parents
- Adaptive set of algorithm parameters (strategy parameters)
- Crossover → Mutation → Selection

# Genetic Programming

- Goal: to learn computer programs from examples (like in machine learning and data mining)

- Main idea:
  represent (simple) *computer programs* in individuals of arbitrary size

- Redefinitions of
  - selection
  - crossover
  - mutation

# Individuals are Program Trees / Parse Trees

- Representation of
  - Arithmetic formulas

$$2 \cdot \pi + \left( (x+3) - \frac{y}{5+1} \right)$$

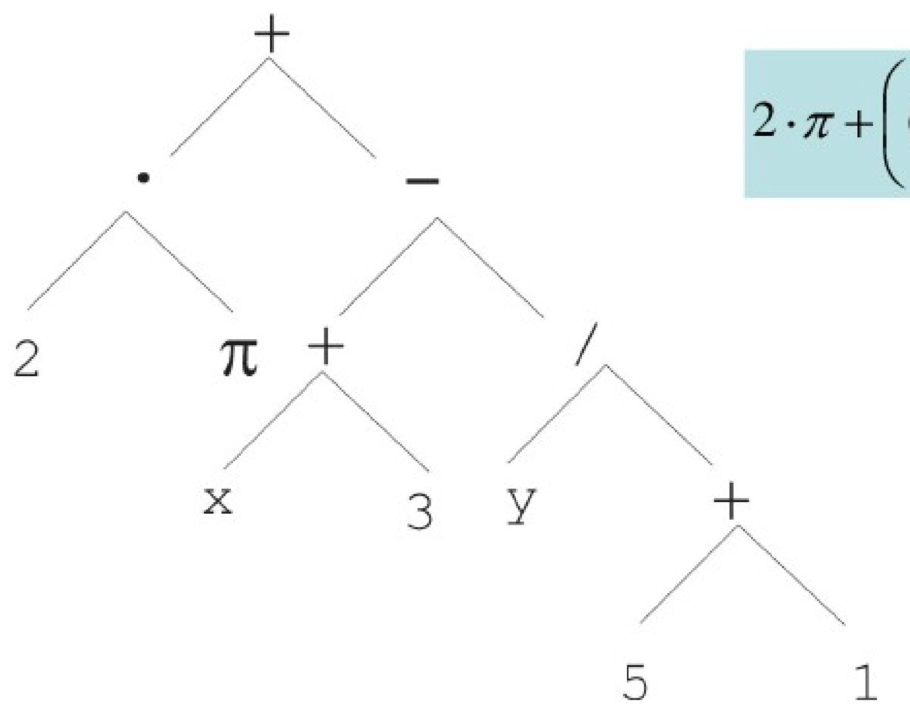
  - Logical formulas

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

  - Computer programs

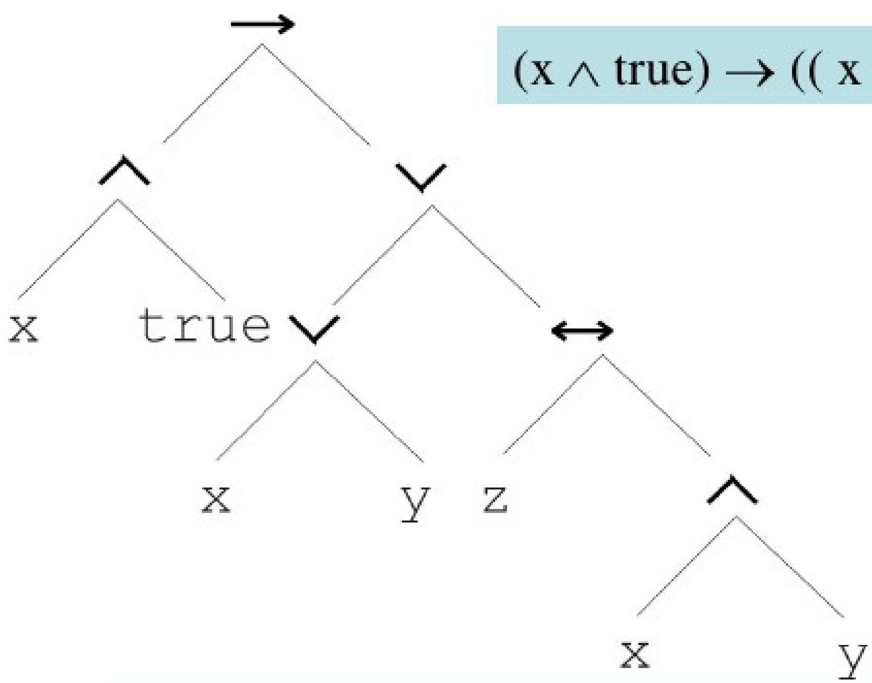
```
i =1;
while (i < 20)
{
        i = i +1
}
```

# Representation of Arithmetic Formula as Tree



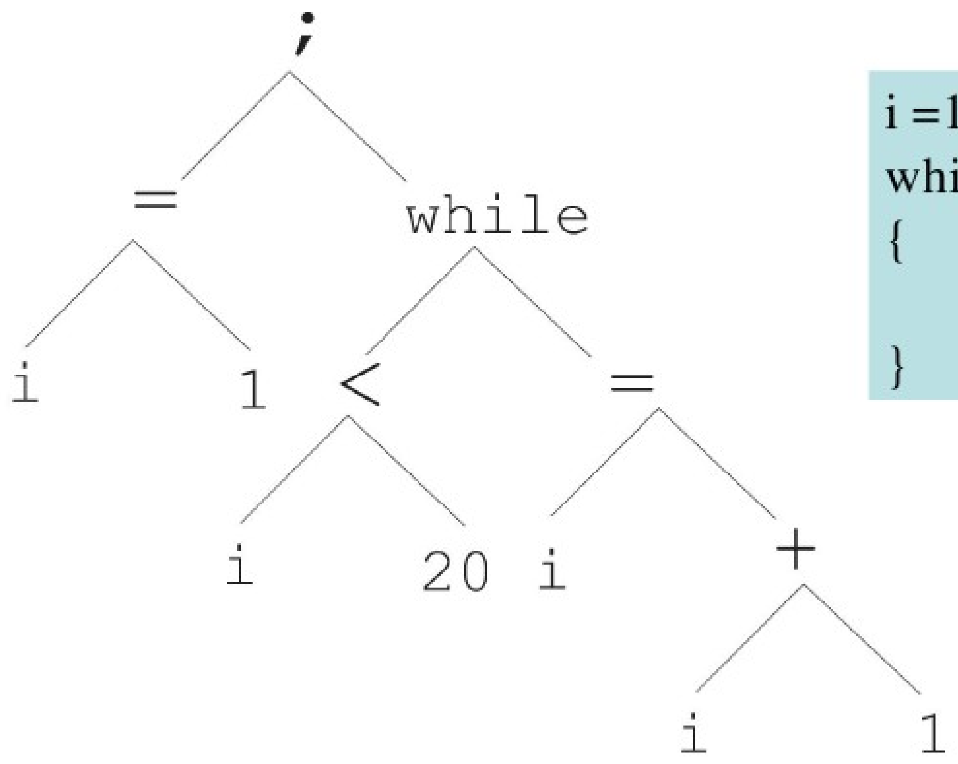

$$2 \cdot \pi + \left( (x+3) - \frac{y}{5+1} \right)$$

# Representation of Logical Formula




$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

# Representation of Computer Programs




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

# Representation

- Trees consisting of:
  - terminals (leaves)
    - constants
    - variables (inputs to the program/formula)
  - functions of fixed arity (internal nodes)

# Considerations in Function Selection

- **Closure:** any function should be well-defined for all arguments

  Example: { *, / } is not closed as division is not well defined if the second argument is 0 → redefine /.

- **Sufficiency:** the function and terminal set should be able to represent a desirable solution

# Evolutionary Cycle

- Fixed population size
- Create a new population by randomly selecting an operation to apply, each of which adds one or two individuals into the new population, starting from one or two fitness proportionally selected individuals:
  - reproduction (copying)
  - one of many crossover operations
  - one of many mutation operations

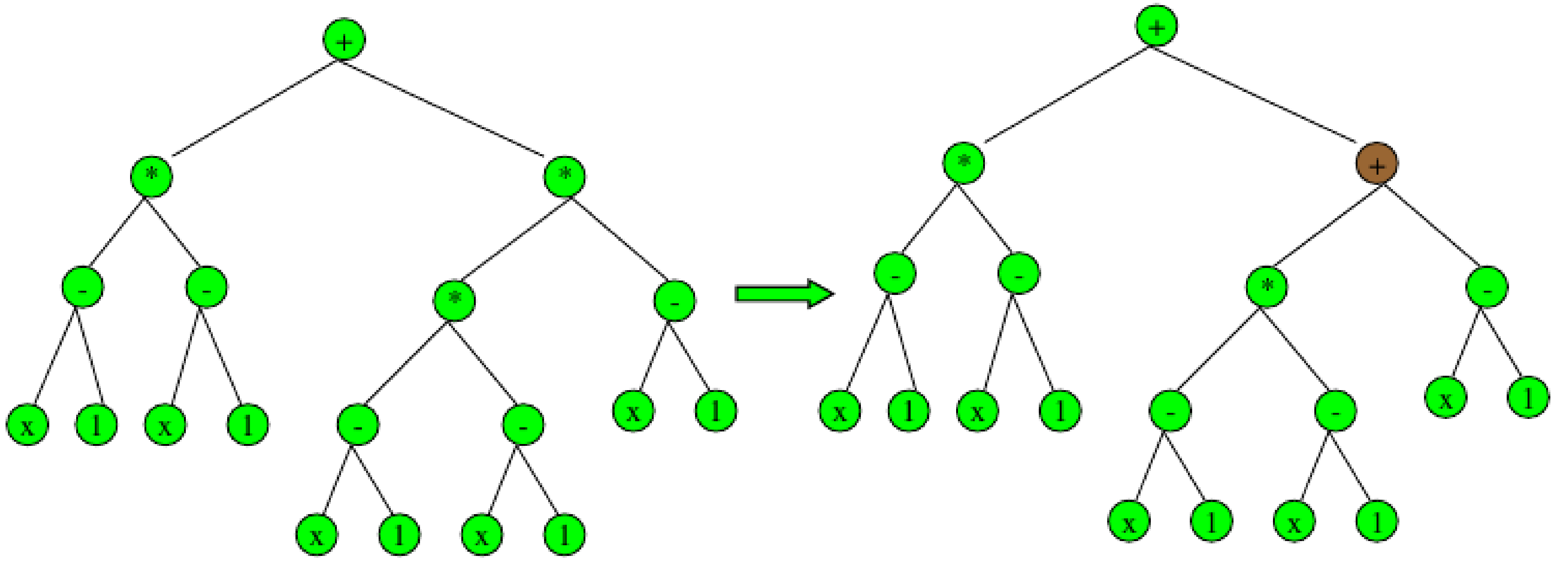
# Initialization

- Given is a maximum depth on trees $D_{max}$
- Full method:
  - for each level < $D_{max}$ insert a node with function symbol (recursively add children of appropriate types)
  - for level $D_{max}$ insert a node with a terminal
- Grow method:
  - for each level < $D_{max}$ insert a node with either a terminal or a function symbol (and recursively add children of appropriate types to these nodes)
  - for level $D_{max}$ insert a node with a terminal
- Combined method: half of the population full, the other grown

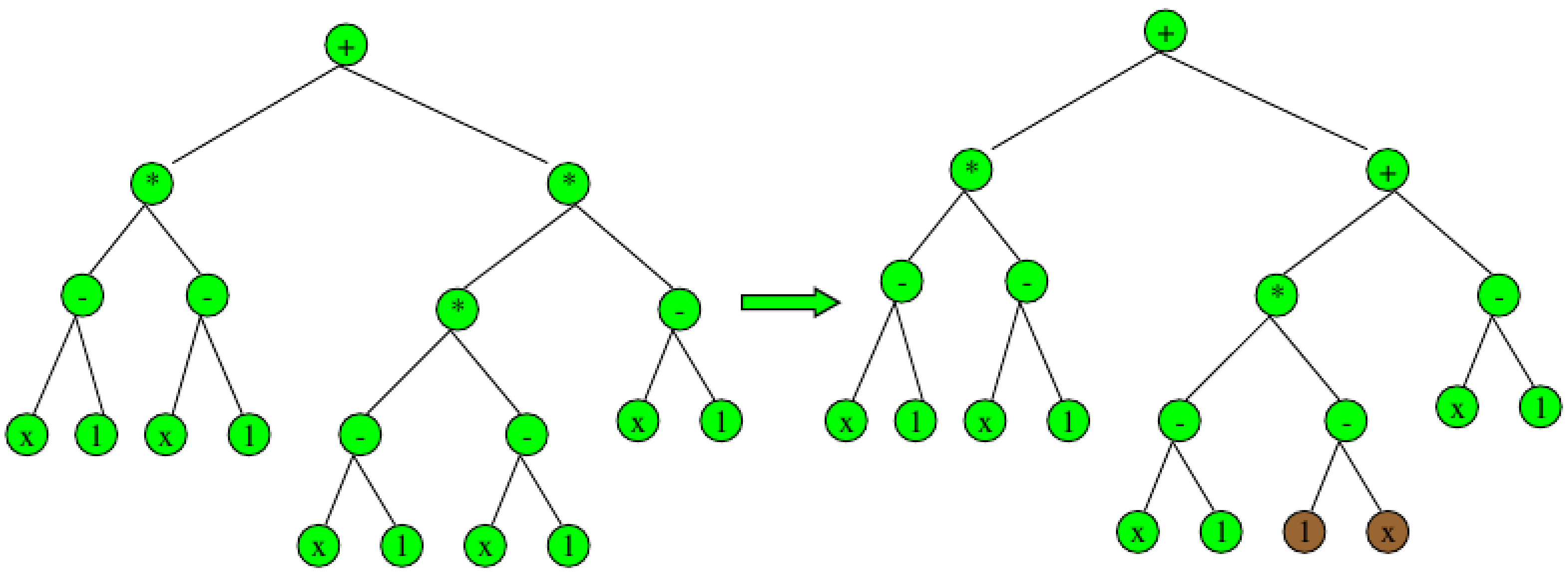
# Mutation

| Operator name | Description |
| --- | --- |
| Point mutation | single node exchanged against random node of same class |
| Permutation | arguments of a node permuted |
| Hoist | new individual generated from subtree |
| Expansion | terminal exchanged against random subtree |
| Collapse subtree | subtree exchanged against random terminal |
| Subtree mutation | subtree exchanged against random subtree |

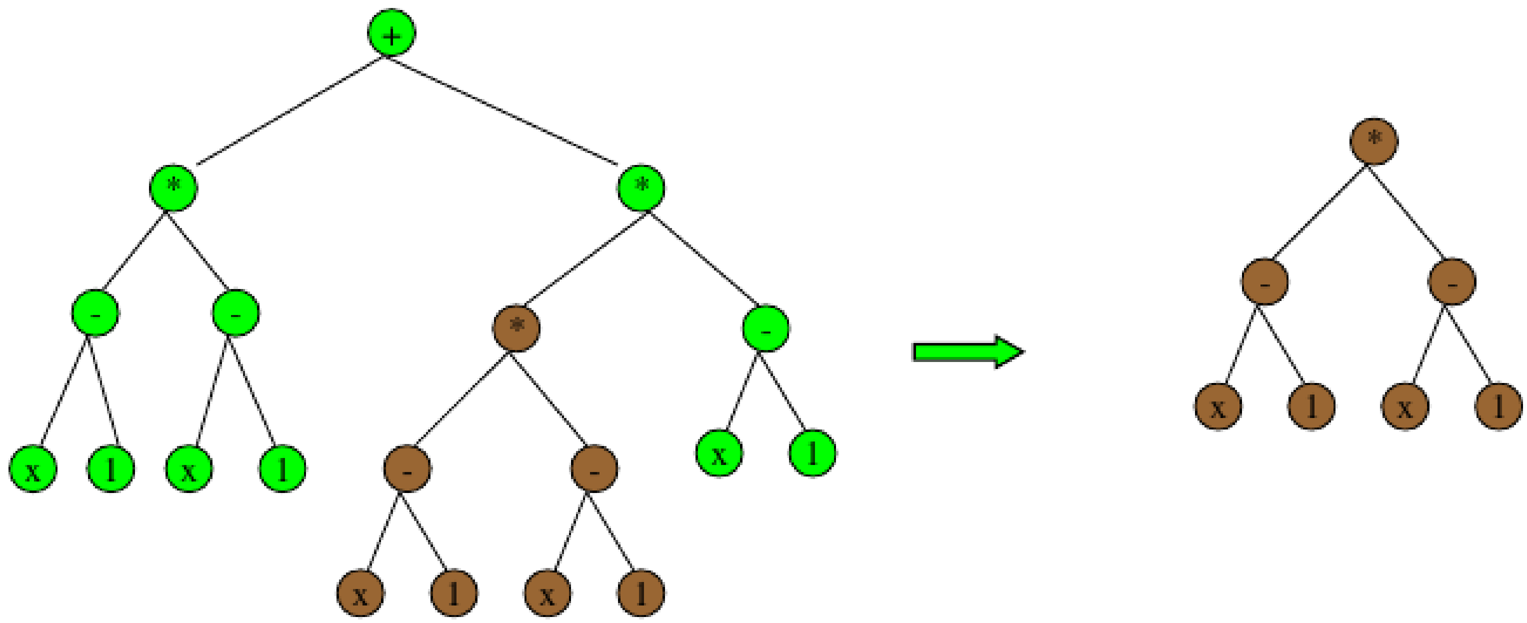
# Point Mutation



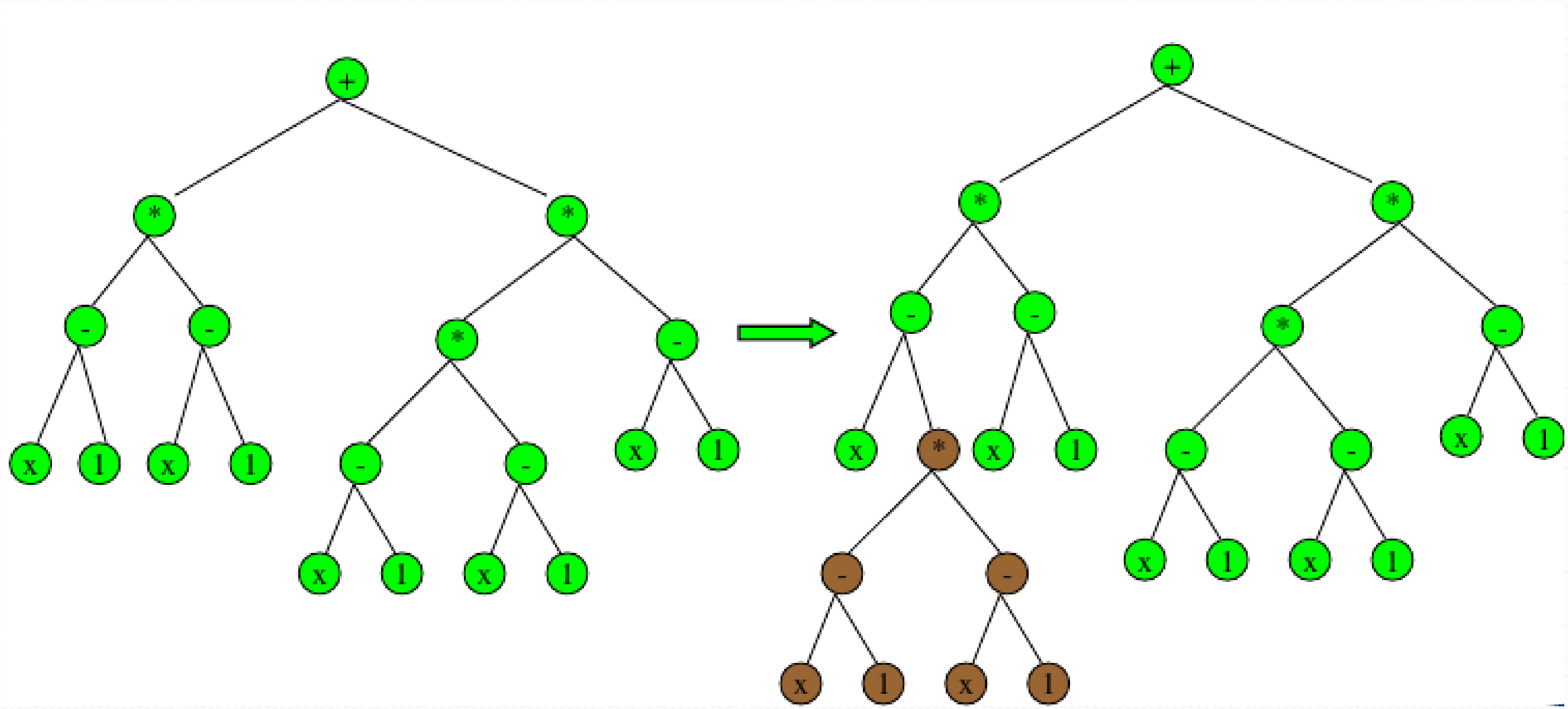
# Permutation



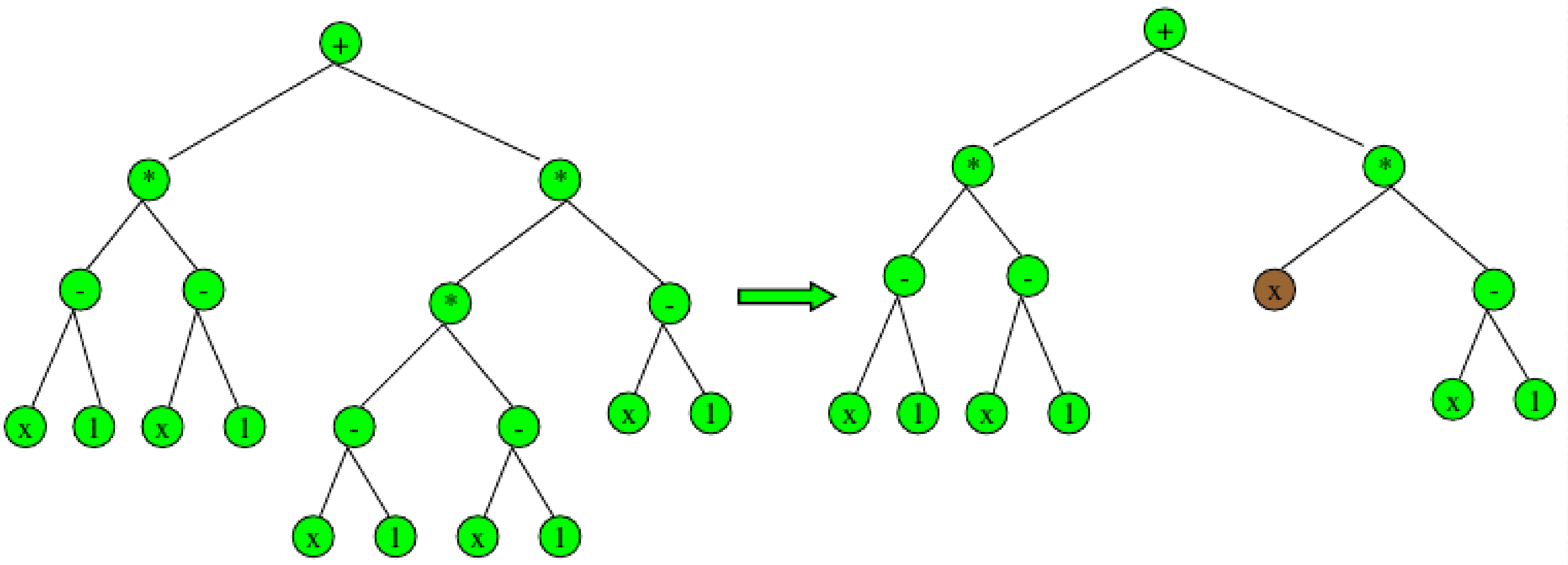
# Hoist



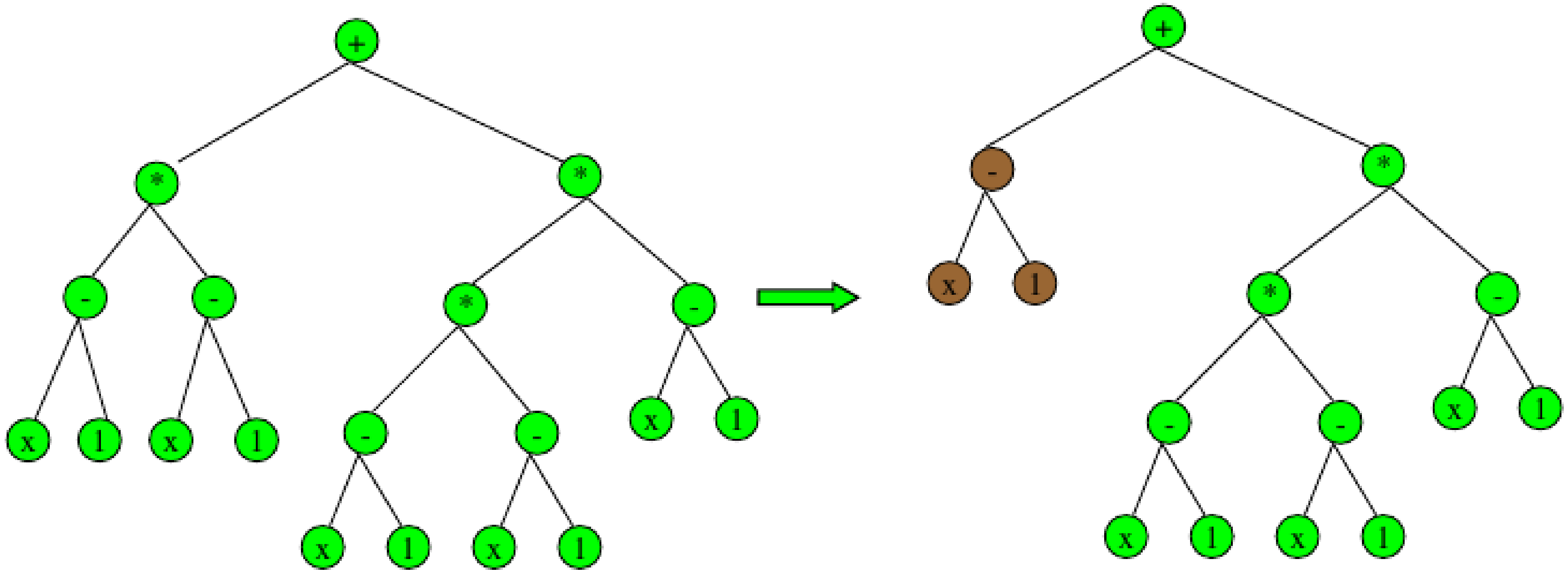
# Expansion Mutation



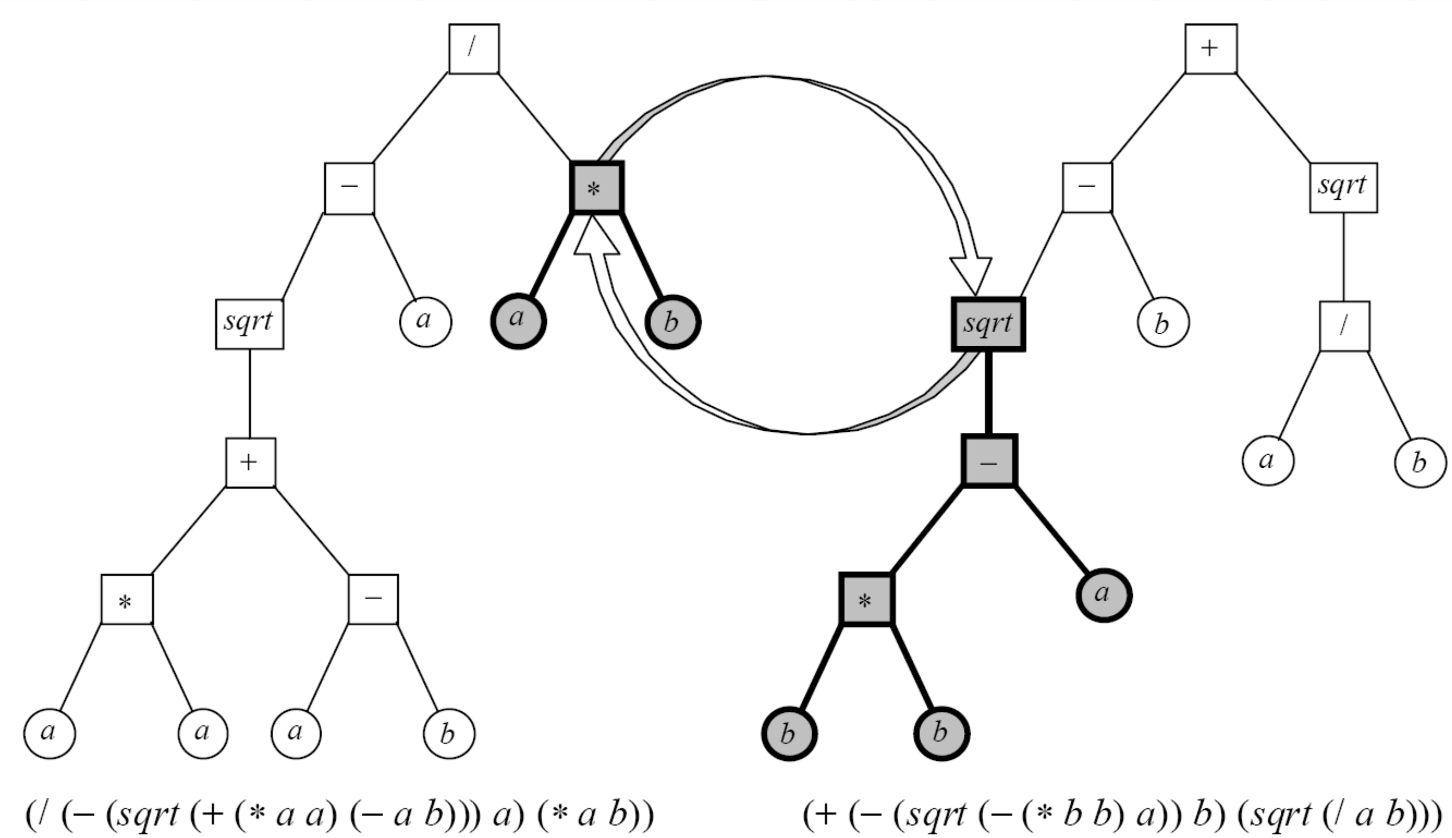
# Collapse Subtree Mutation



# Subtree Mutation

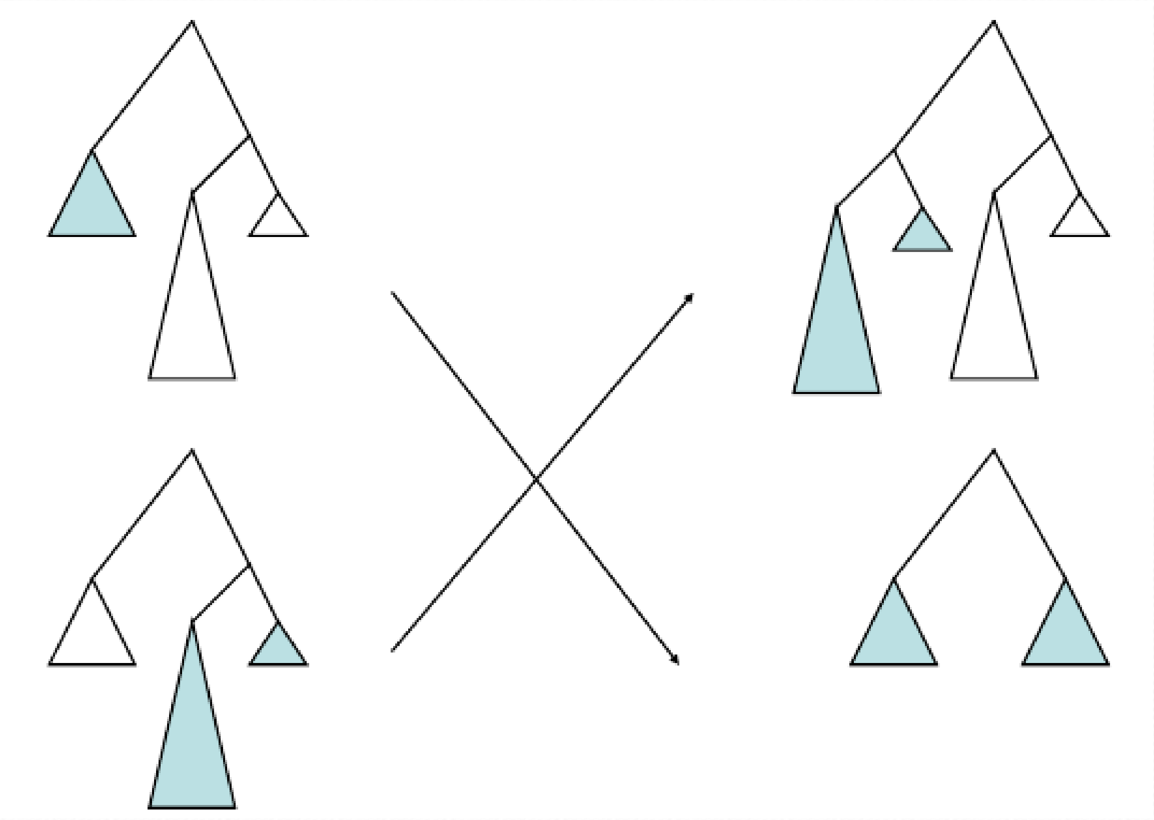


# Crossover



/
−
*
sqrt
a
a
b
+
*
−
a
a
a
b
+
−
sqrt
sqrt
b
/
−
a
b
*
a
b
b
b
(/ (− (sqrt (+ (* a a) (− a b))) a) (* a b))
(+ (− (sqrt (− (* b b) a)) b) (sqrt (/ a b)))

# Self-Crossover



# Bloat

- “Survival of the fattest”, i.e. the tree sizes in the populations increase over time

- Countermeasures:
  - simplification
  - penalty for large trees
  - hard constraints on the size of trees resulting from operations

# Editing Operator

- An operation that simplifies expressions
- Examples:
  - X AND X → X
  - X OR X → X
  - NOT(NOT(X)) → X
  - X + 0 → X
  - X . 1 → X
  - X . 0 → 0
  - ....

# Example – Symbolic Regression Pythagorean Theorem

Not (necessarily) linear

Negnevitsky 2004

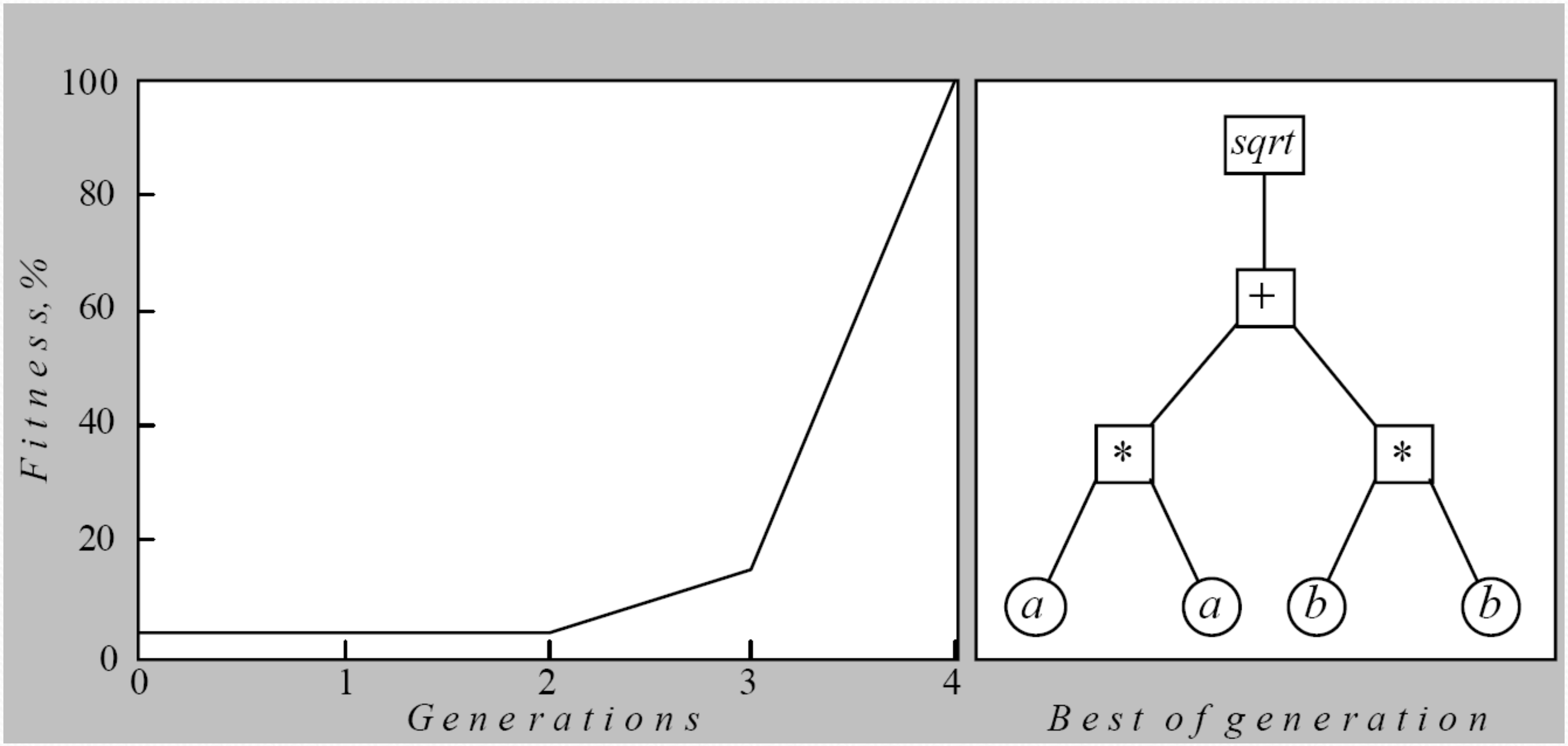
Underlying function: $c = \sqrt{a^2 + b^2}$

Fitness cases:

| Side $a$ | Side $b$ | Hypotenuse $c$ | Side $a$ | Side $b$ | Hypotenuse $c$ |
|---|---|---|---|---|---|
| 3 | 5 | 5.830952 | 12 | 10 | 15.620499 |
| 8 | 14 | 16.124515 | 21 | 6 | 21.840330 |
| 18 | 2 | 18.110770 | 7 | 4 | 8.062258 |
| 32 | 11 | 33.837849 | 16 | 24 | 28.844410 |
| 4 | 3 | 5.000000 | 2 | 9 | 9.219545 |

Language elements: +, -, *, /, sqrt, *a*, *b*

# Results

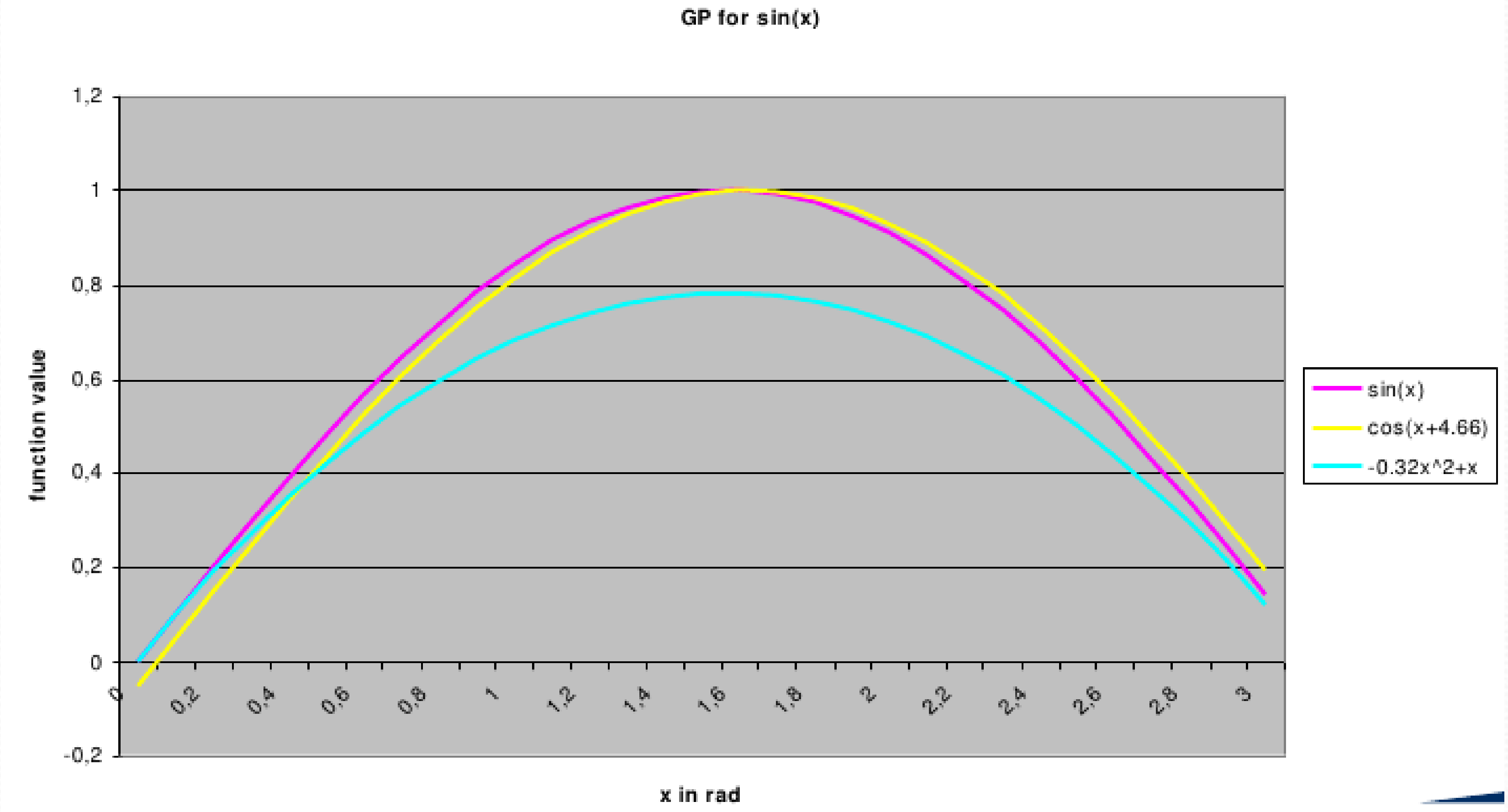

100
80
60
40
20
0
Fitness, %
0
1
2
3
4
Generations
sqrt
+
*
*
a
a
b
b
Best of generation

# Example – Symbolic Regression Approximation of sin(x)

- **Given** examples (x,sin(x)) with x in {0,1,…,9}
- **Find** a good approximation of sin(x)

| Function Sets | Result | Generation | Error (final) |
|---|---|---|---|
| $F_1$: { +, -, *, /, sin } | $sin(x)$ | 0 | 0.00 |
| $F_2$: { +, -, *, /, cos } | $cos(x + 4.66)$ | 12 | 0.40 |
| $F_3$: { +, -, *, / } | $-0.32\,x^2 + x$ | 29 | 1.36 |

# Example – Symbolic Regression Approximation of sin(x)



# GAs vs. GP

Genetic algorithms

- Chromosomes represent coded solutions
- Fixed length chromosomes
- A small set of well-defined genetic operators
- Conceptually simple
- Fixed order of operators

Genetic programming

- Chromosomes represent executable code
- Variable length chromosomes
- More complex genetic operators required
- Conceptually complex
- Order of operators not fixed