UNIVERSIDAD COMPLUTENSE MADRID

Adaptive and Bioinspired Systems Research Group

Genetic Programming based on Grammars

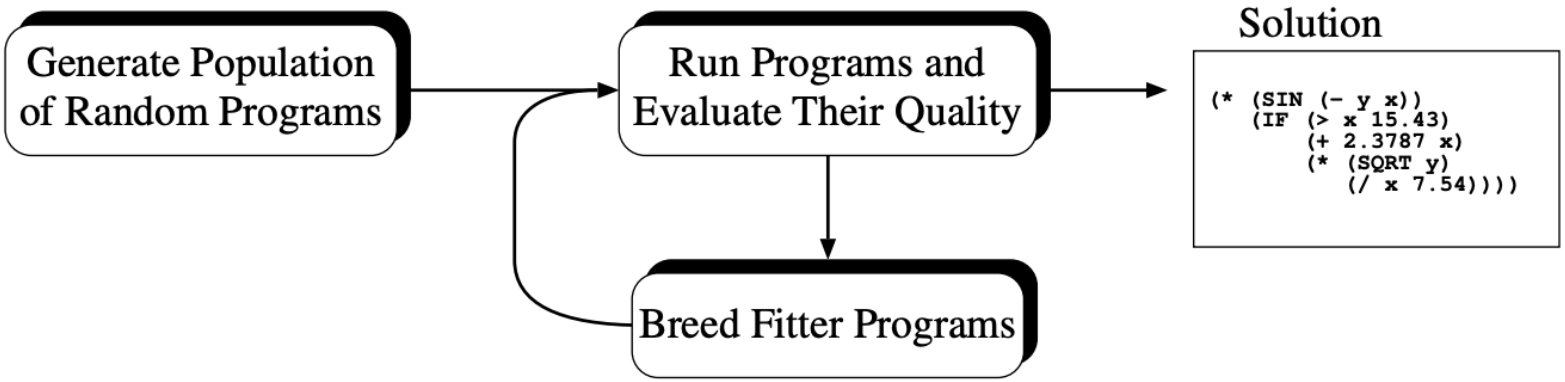
Iñaki Hidalgo

Overview

- Genetic Programming
- Grammars
- Genetic Programming based on Grammars
  - Tree based GGGP
  - Grammatical Evolution (GE)
  - Structured GE
    - Static
    - Dynamic
  - Tree-Based Grammatical Evolution with Non-Encoding Nodes
- Some Applications
  - Symbolic Regressio
  - Classification
- Advanced Topics
  - Lexicase selection
  - Probabilistic GE
- Conclusions

Genetic Programming (GP)

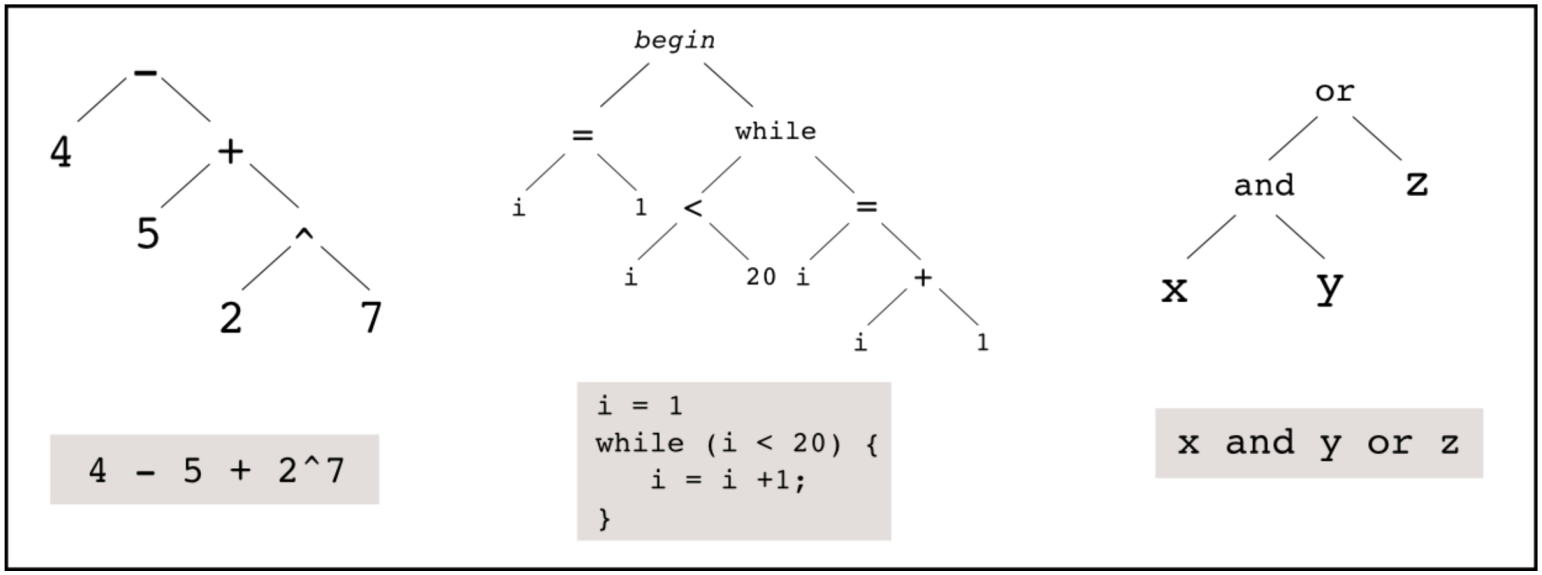
- Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance.
- GP is a systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done.



Poli, Riccardo ; Langdon, William & Mcphee, Nicholas (2008). A Field Guide to Genetic Programming.

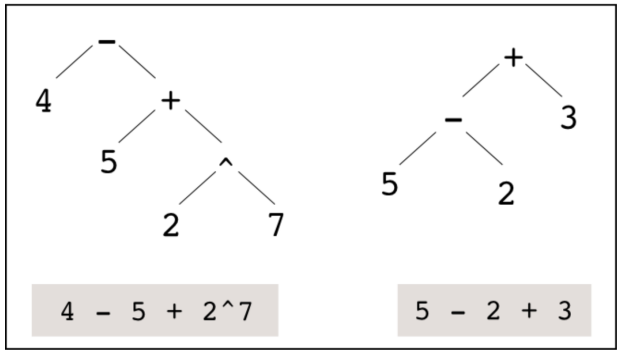
Genetic Programming (GP)

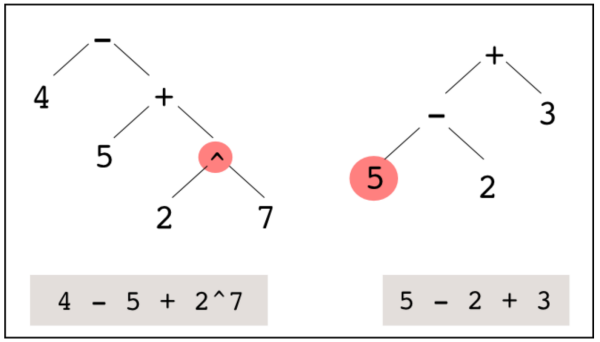
- Individuals are represented as syntax trees
  - Computer Programs
  - Arithmetic expressions
  - Logical forms
  - ….

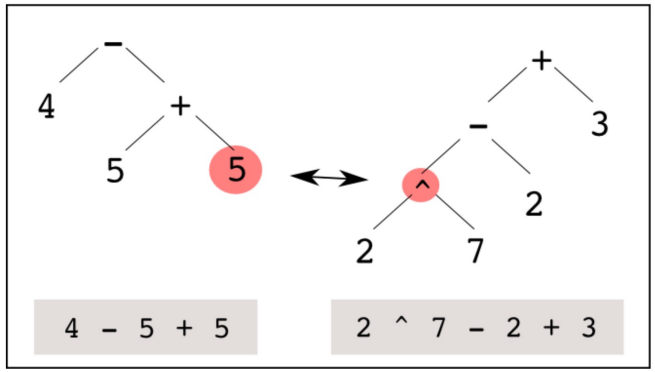


Genetic Programming (GP)

- Recombination
  - Interchanging sub-trees

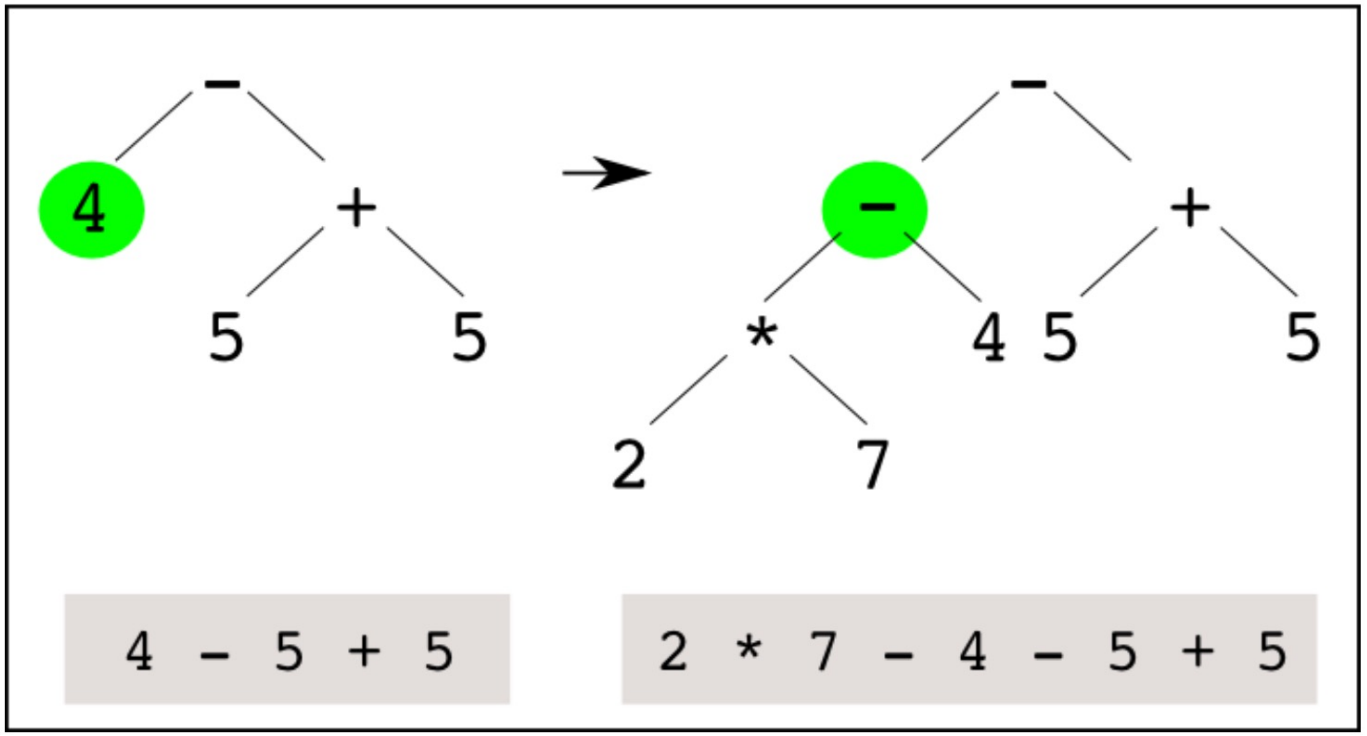






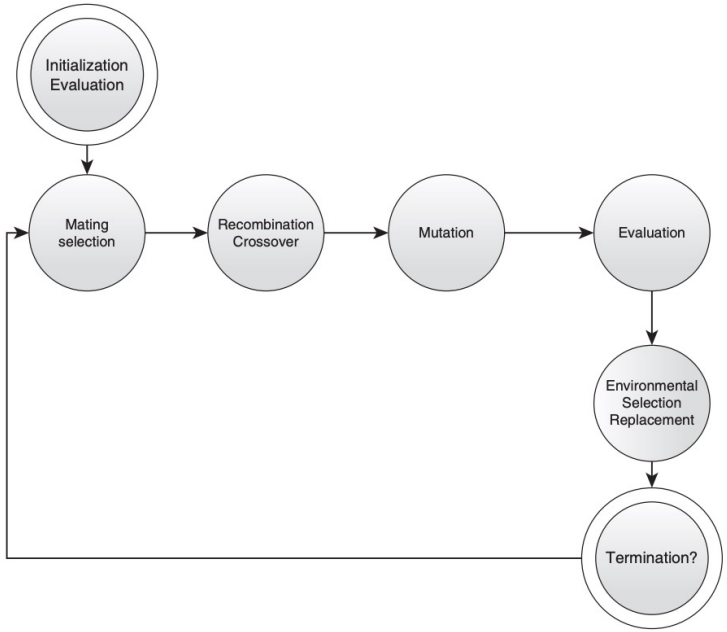
Genetic Programming (GP)

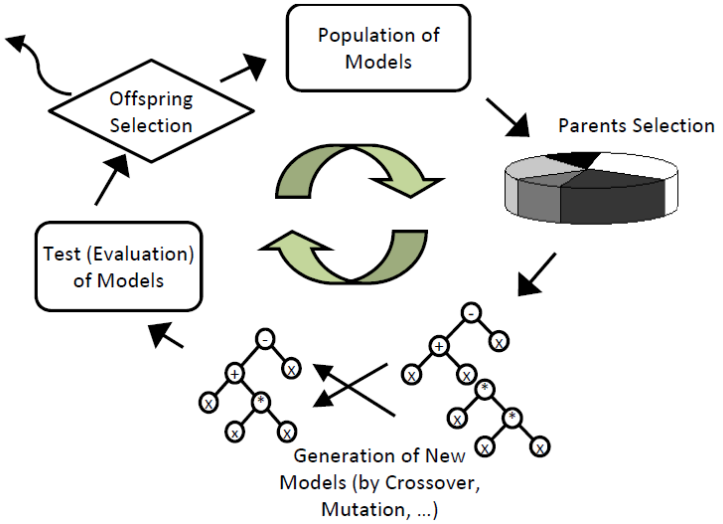
- Mutation



Genetic Programming (GP)

- Remember !!
  - It is an Evolutionary Algorithm!!
  - Population Based



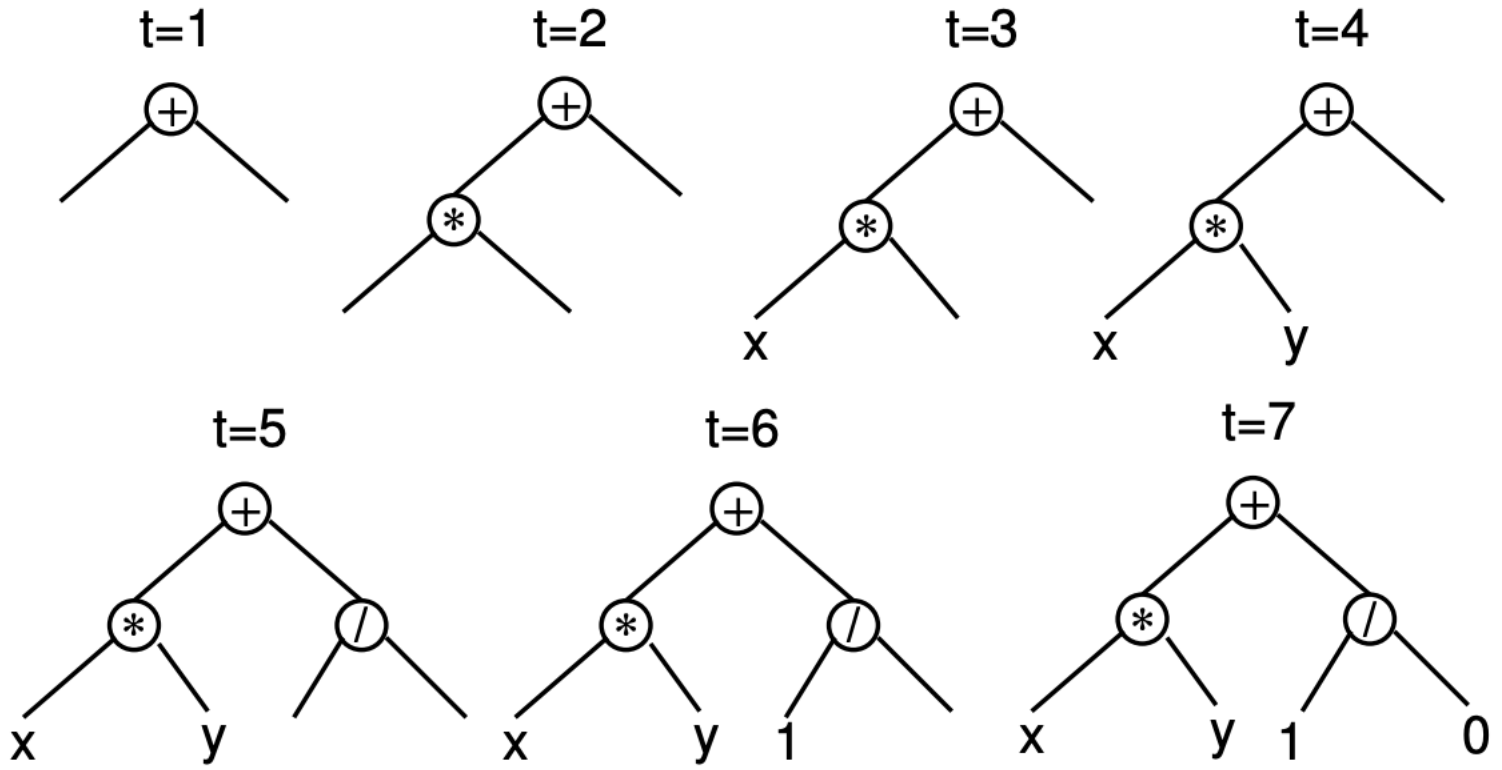


Genetic Programming (GP)

- Depth of a node
  - The number of edges that need to be traversed to reach the node starting from the tree's root node
- Initialization
  - Important Maximum Depth
    - Complexity
    - Efficiency
  - Full Method
    - it generates full trees, i.e. all leaves are at the same Depth
    - nodes are taken at random from the function set until the maximum tree depth is reached. Beyond that depth, only terminals can be chosen.
    - May be limited
  - Grow method
    - More varied
    - Nodes are selected from the whole primitive set (i.e., functions and terminals) until the depth limit is reached.
  - Ramped half-and-half
    - Half the initial population is constructed using full and half is constructed using grow.
    - Using a range of depth limits (ramped)

Genetic Programming (GP)

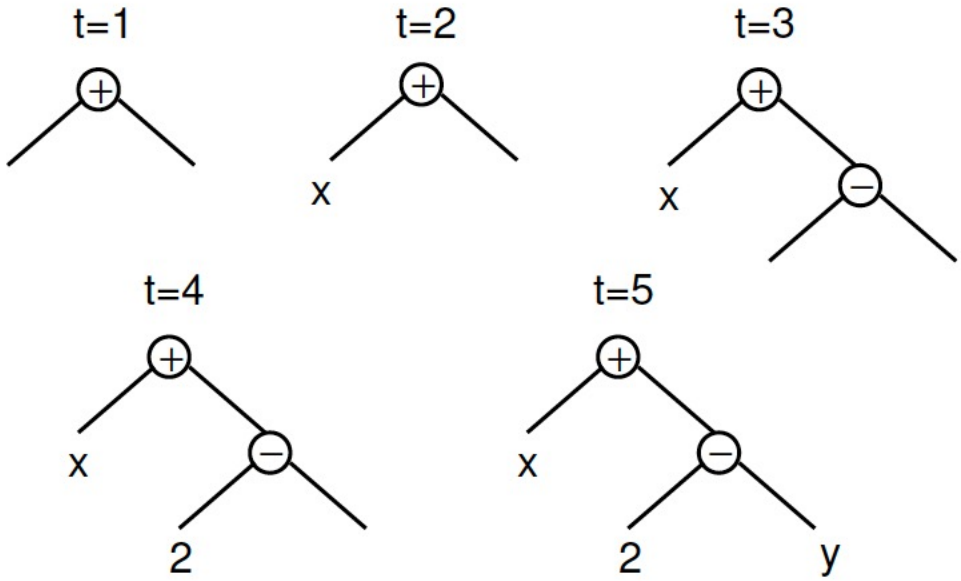
- Full method



Poli, Riccardo ; Langdon, William & Mcphee, Nicholas (2008). A Field Guide to Genetic Programming.

Genetic Programming (GP)

- Grow method



Poli, Riccardo ; Langdon, William & Mcphee, Nicholas (2008). A Field Guide to Genetic Programming.

Bloat

- Uncontrolled growing of trees
  - May be Fittest solutions or not

Genetic Programming (GP)

- What it the terminal set?
- What is the function set?
- What is the fitness function?
- What parameters will be used for controlling the run?
  - Selection
  - Initialization
  - Crossover
  - mutation
- What will be the termination criterion
- What is expected to be the result of the run?

Grammars

- Context Free Grammar
  - Formal Grammar
  - Composed by production rules
    - A → b
    - A is a non terminal symbol
    - b is a terminal symbol
  - The non-terminal A can always be replaced by b , irrespective of the context
- Describe programming languages
- Restrictions
- Chomsky normal form
  - All of its production rules are of the form
    - A→BC
    - A →a

Grammar-based Constraints

- Each line in a grammar is a production rule.
- Elements that cannot be rewritten are known as the terminals of the grammar
  - Not to be confused with the terminals in the primitive set of a GP
- Symbols that appear on the left-hand-side of a rule are known as non-terminal symbols.
- Non terminals may appear on the left-hand-side or on the right-hand-side

$$
\begin{aligned}
\mathit{tree} &::= E \times \sin(E \times t) \\
E &::= \mathtt{var} \mid (E\ \mathtt{op}\ E) \\
\mathtt{op} &::= + \mid - \mid \times \mid \div \\
\mathtt{var} &::= x \mid y \mid z
\end{aligned}
$$

BNF grammars

- Backus Naur Form (BNF) is a notation for expressing the grammar of a language in the form of production rules.
- BNF grammars consist of
  - terminals
    - items that can appear in the language.
  - non-terminals
    - can be expanded into one or more terminals and non-terminals.
- A BNF grammar is made up of the tuple N, T, P, S

$$N = \{expr, op, pre\_op\}$$

$$T = \{Sin, Cos, Tan, Log, +, -, /, *, X, ()\}$$

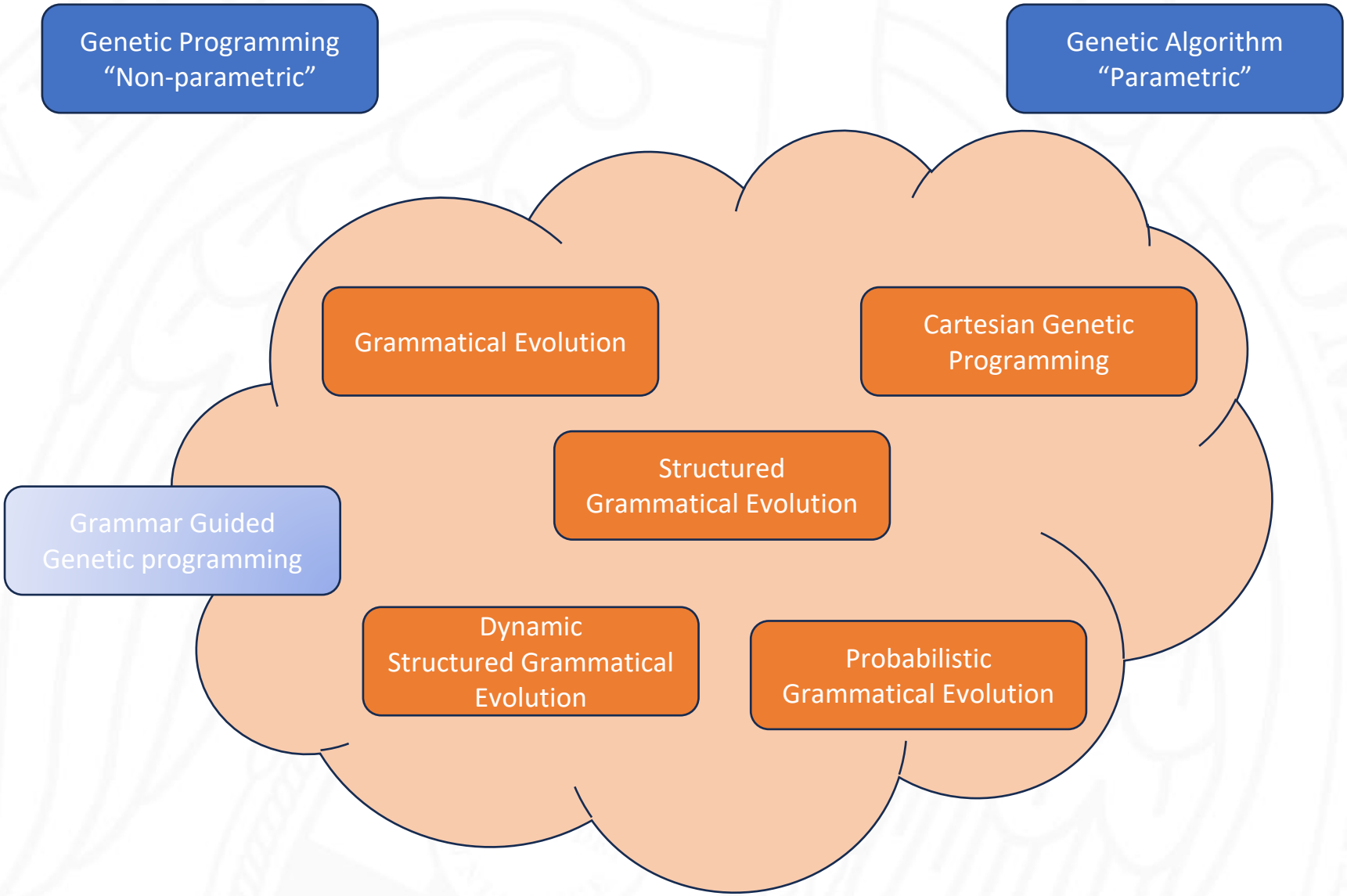
$$S = < expr >$$

```
(1) <expr> ::= <expr> <op> <expr>      (A)
             | ( <expr> <op> <expr> )  (B)
             | <pre-op> ( <expr> )     (C)
             | <var>                   (D)

(2) <op> ::= + (A)
           | - (B)
           | / (C)
           | * (D)

(3) <pre-op> ::= Sin (A)
               | Cos (B)
               | Tan (C)
               | Log (D)

(4) <var> ::= X
```

Models

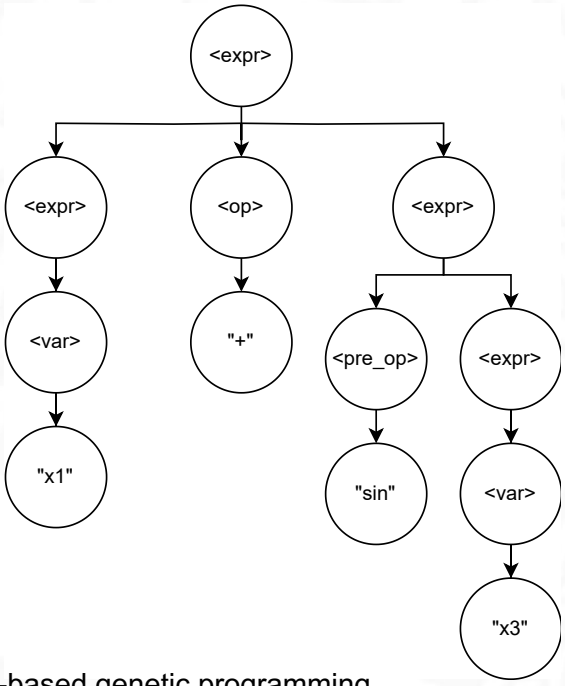
- Parametric
  - $Y= a* x_1 + b*x_2+ c*x_3+ d*x_4^2+e$
  - Developed by experts
  - Dangerous
- Non Parametric
  - Unknown structure
  - Data-based
  - Symbolic regression
    - Obtain an expresión that best represents the data as a function of the variables.
  - Huge search spaces
- Intermediate
  - Well known parts + Unknown parts
  - Constraints

Genetic Programming based on Grammars



Tree-based Grammar Guided GP

- Context-Free Grammar GP (CFG-GP)
- Individuals are derivation trees whose creation is guided by the grammar
- Nodes contain a symbol and a pointers to child nodes
- Root node is the first non-terminal symbol in the grammar
- Example



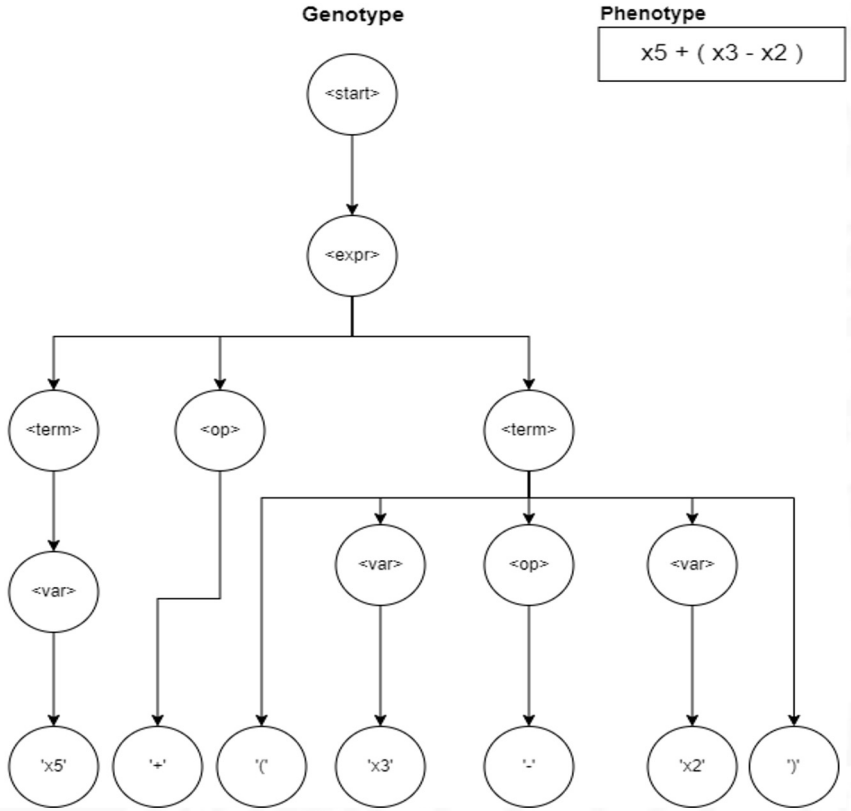
Whigham, P. (1999) Grammatically-based genetic programming

Tree-based Grammar Guided GP

- One-to-one genotype to phenotype correlation
- Sub-tree cross-over
- Regenerative sub-tree mutation

Grammar

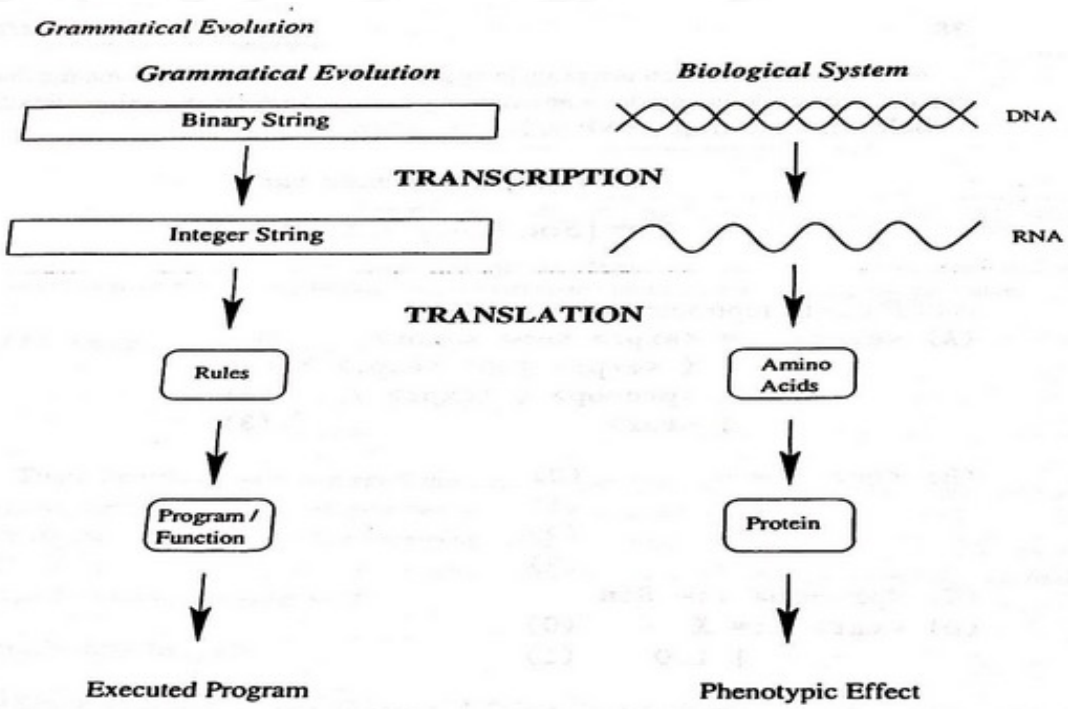
```
<start> ::= <expr> | <term>
<expr> ::= <term> <op> <term>
<term> ::= ( <var> <op> <var> ) | <var>
<var> ::= x1 | x2 | x3 | x4 | x5
<op> ::= + | - | *
```

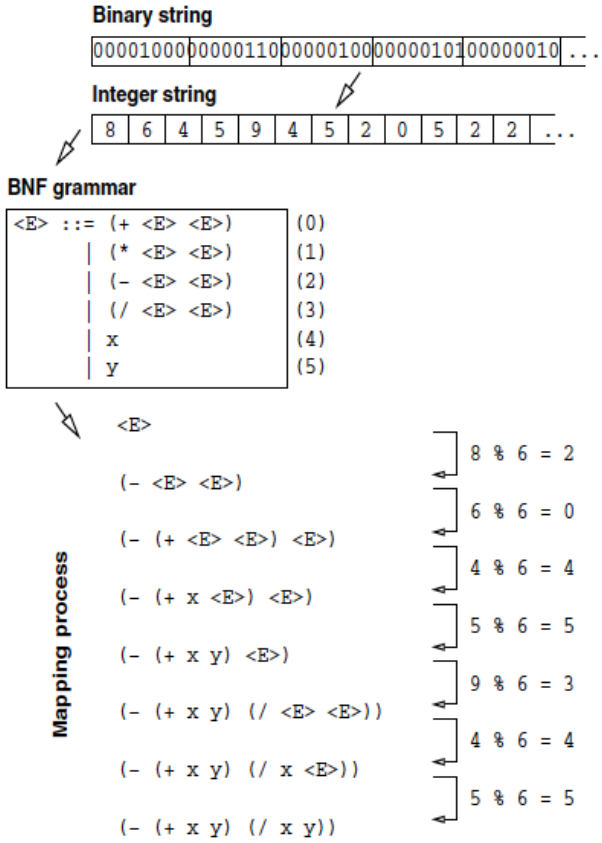


Tree-based Grammar Guided GP

- Whigham et al. introduced tree-based GGGP to support more expressive search space definitions via a CFG.
  - Individuals are generated through the expansion of the grammar, using its production rules.
  - Compared to standard GP, tree-based GGGP differs in the mutation and recombination operator, which the CFG restricts.
  - The selection of compatible nodes in trees is traditionally considered more computationally expensive (O(log n)) than selecting an element in a list (O(1)).
  - Additionally, selecting a target node expanded from a non-terminal symbol is equiprobable.
  - It is more common that the selected node is not a terminal, especially in recursive grammars.
    - Therefore, individuals can be pretty deep which may result in higher memory consumption and bloat in regression problems.

Biology System vs. Grammatical Evolution





Ryan, C., Collins, J. J., & Neill, M. O. (1998, April). Grammatical evolution: Evolving programs for an arbitrary language. In European Conference on Genetic Programming (pp. 83-96). Springer Berlin Heidelberg.

Grammatical Evolution (GE)

- Use a grammar to define the structure of individuals
  - Specifically, context-free grammar, G, defined as

$$G = \{N, T, S, P\}$$

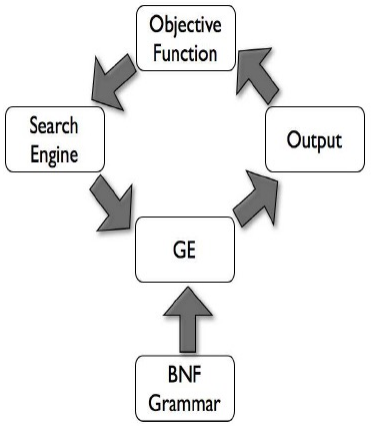
where:

N: set of non-terminal symbols

T: set of terminal symbols

S: starting rule, a.k.a. axiom

P: set of production rules, i.e., $A ::= B$ with B (




```
<expr> ::= <expr> <op> <expr> |  <pre_op> <expr> | <var>
<var> ::= x1 | x2 | x3 | x4
<op> ::= + | -
<pre_op> ::= log | sin | cos
```

Ryan, C., Collins, J. J., & Neill, M. O. (1998, April). Grammatical evolution: Evolving programs for an arbitrary language. In European Conference on Genetic Programming (pp. 83-96). Springer Berlin Heidelberg.

Grammatical Evolution (GE)

- Linear representation of a GGGP
- List of values with predetermined size decoded using the grammar
- Example [3, 2, 8, 4, 1, 7, 5, 14]

$$Rule = c\%r$$

S → <expr> ::= <expr> <op> <expr> | <pre_op> <expr> | <var>
<var> ::= x1 | x2 | x3 | x4
<op> ::= + | -
<pre_op> ::= log | sin | cos

$$\langle expr \rangle \langle op \rangle \langle expr \rangle$$

Grammatical Evolution (GE)

- Linear representation of a GGGP
- List of values with predetermined size decoded using the grammar
- Example [3, 2, 8, 4, 1, 7, 5, 14]

<expr> ::= <expr> <op> <expr> | <pre_op> <expr> | <var>
<var> ::= x1 | x2 | x3 | x4
<op> ::= + | -
<pre_op> ::= log | sin | cos

$$x1 + \sin(x3)$$

Grammatical Evolution (GE)

- Linear representation of a GGGP
- List of values with predetermined size decoded using the grammar
- Example [3, 2, 8, 4, 1, 7, 5, 14]

$$Rule = c\%r$$

<expr> ::= <expr> <op> <expr> | <pre_op> <expr> | <var>
<var> ::= x1 | x2 | x3 | x4
<op> ::= + | -
<pre_op> ::= log | sin | cos

⟨expr⟩ ⟨op⟩ ⟨expr⟩

Grammatical Evolution (GE)

- Linear representation of a GGGP
- List of values with predetermined size decoded using the grammar
- Example [3, 2, 8, 4, 1, 7, 5, 14]

$$Rule = c\%r$$

<expr> ::= <expr> <op> <expr> | <pre_op> <expr> | <var>
<var> ::= x1 | x2 | x3 | x4
<op> ::= + | -
<pre_op> ::= log | sin | cos

$\langle var \rangle \langle op \rangle \langle expr \rangle$

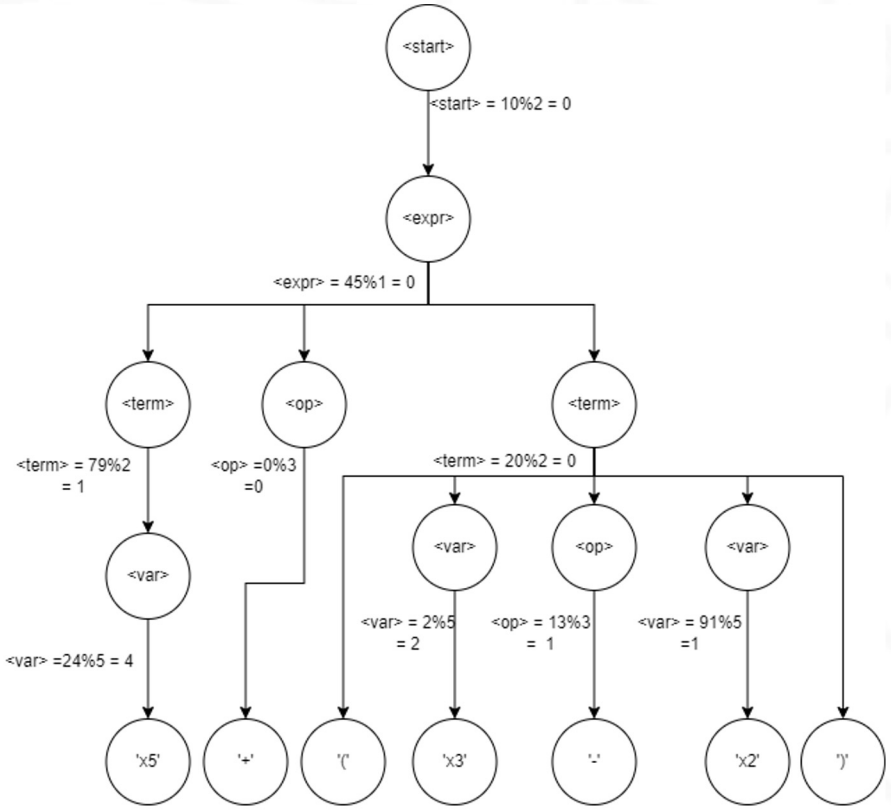
Grammatical Evolution (GE)

Grammar

```
<start> ::= <expr> | <term>
<expr> ::= <term> <op> <term>
<term> ::=  ( <var> <op> <var> ) | <var>
<var> ::= x1 | x2 | x3 | x4 | x5
<op> ::= + | - | *
```

Individual's Genotype

[10,45,79,24,0,20,2,13,91]



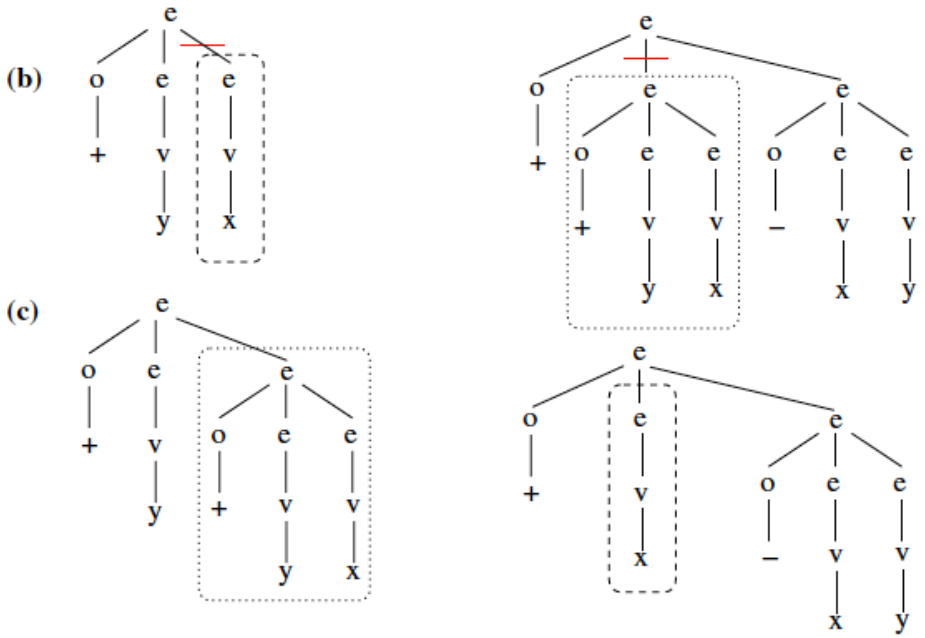
Genotype

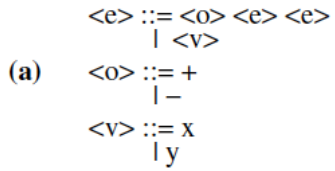
[10,45,79,24,0,20,2,13,91]

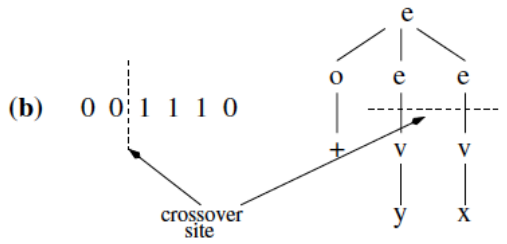
Phenotype

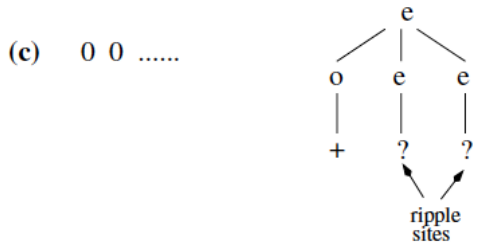
x5 + (x3 - x2)

Crossover in GP vs GE





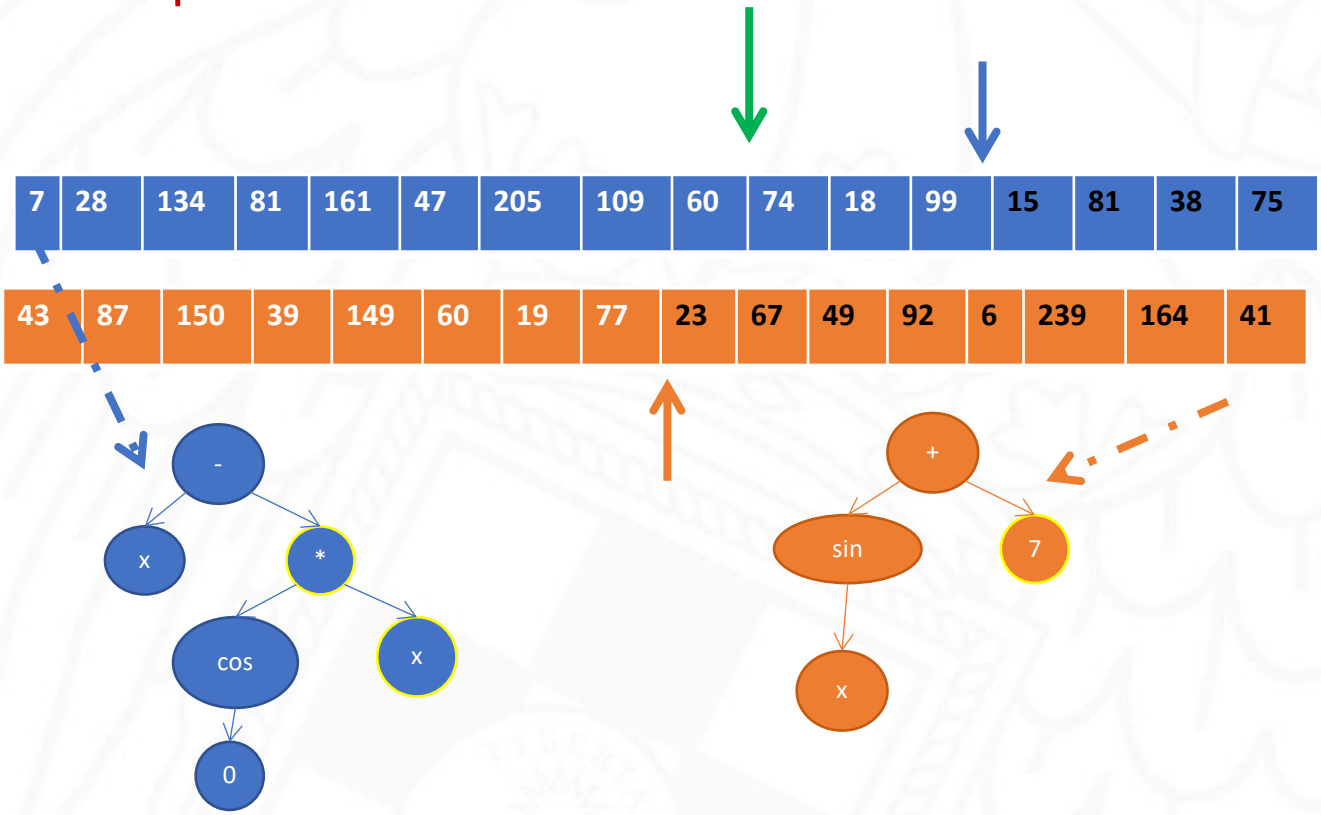




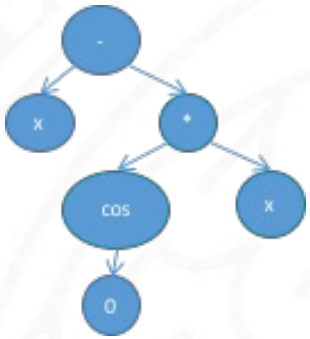
O'neill, M., Ryan, C., Keijzer, M., & Cattolico, M. (2003). Crossover in grammatical evolution. Genetic programming and evolvable machines, 4(1), 67-93.

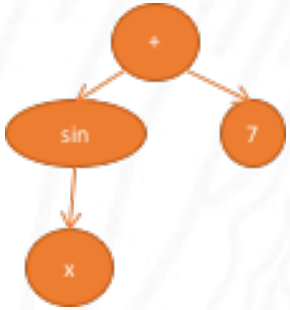
Grammatical Evolution (GE)

One point crossover

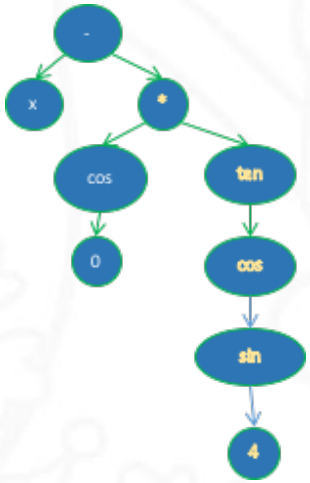


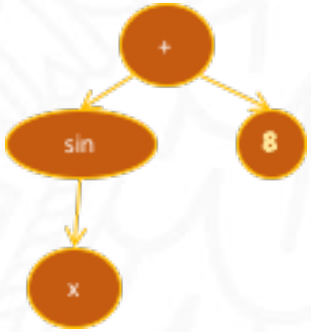
Grammatical Evolution (GE)





Crossover

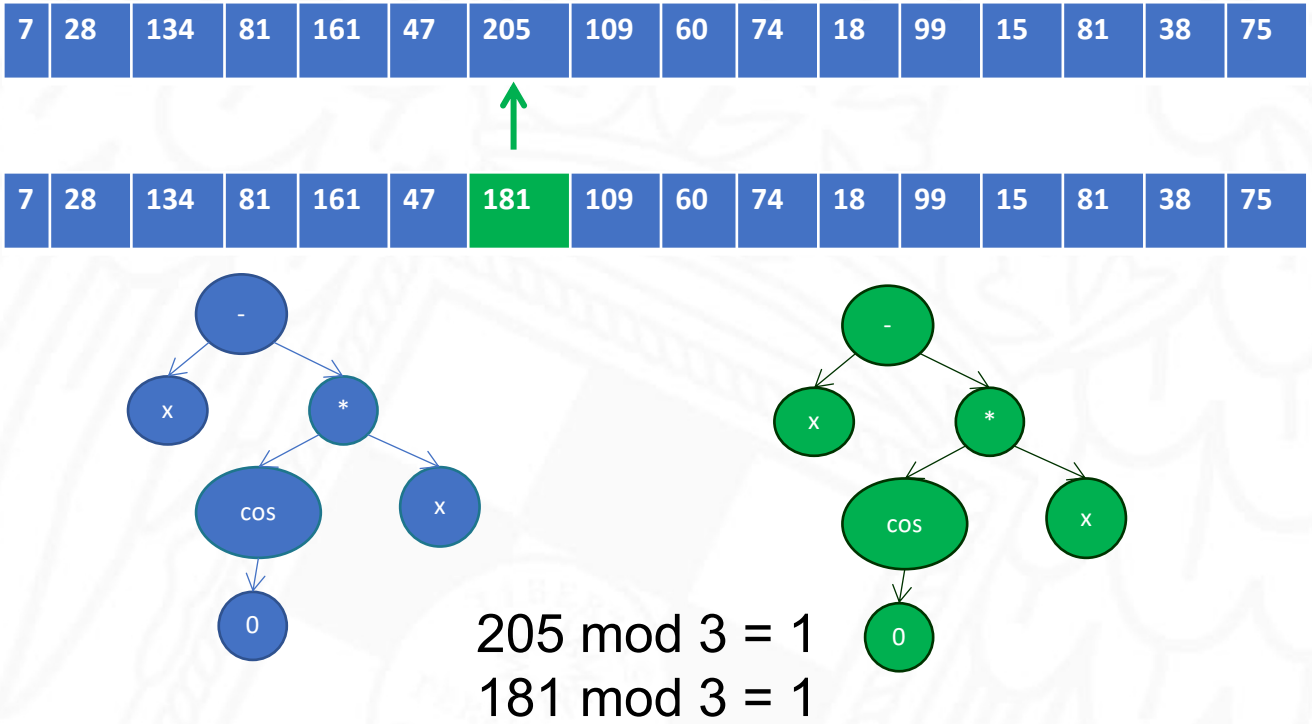




Grammatical Evolution (GE)

Neutral Mutation

- No phenotype changes, but important for the evolutionary process?

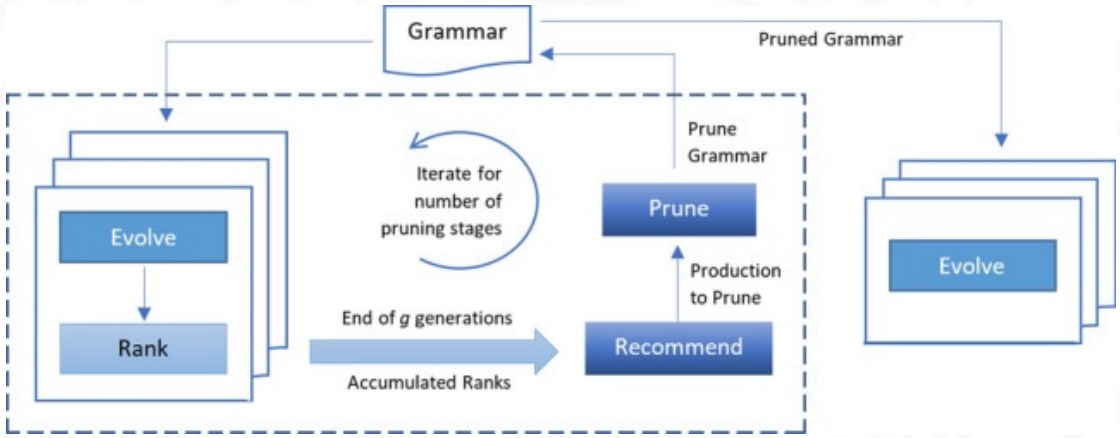


Grammatical Evolution (GE)

Neutral Mutation

Grammatical Evolution (GE)

- Wrapping
  - Reuse the chromosome if necessary to finish the decoding process
  - Some authors do not recommend it
    - Fix a maximum number of wrappings
    - Discard the individual
    - Assign a huge fitness
- Pruning



Ali MS, Kshirsagar M, Naredo E, Ryan C. Dynamic Grammar Pruning for Program Size Reduction in Symbolic Regression. SN Comput Sci. 2023;4(4):402. doi: 10.1007/s42979-023-01840-y. Epub 2023 May 17. PMID: 37214587; PMCID: PMC10192180.

Structured GE (SGE)

- List of lists
- Each list associated to a non-terminal in the grammar
- SGE genotype with the same phenotype in previous grammar

<expr>	<var>	<op>	<pre_op>
[[0, 2, 1, 2] ,	[0, 2] ,	[0],	[1]]

```
<expr> ::= <expr> <op> <expr> |  <pre_op> <expr> | <var>
<var> ::= x1 | x2 | x3 | x4
<op> ::= + | -
<pre_op> ::= log | sin | cos
```

- Pros
  - Alleviates redundancy in GE
  - More locality in crossover
- Dynamic SGE
  - Generates the list on-the-fly up to the specific individual needs
  - Only valid individual in population
  - Mitigate bloating

(2) Lourenço, N., Pereira, F., Costa, E.: SGE (2015) A structured representation for grammatical evolution

Structured Grammatical Evolution (SGE)

Grammar

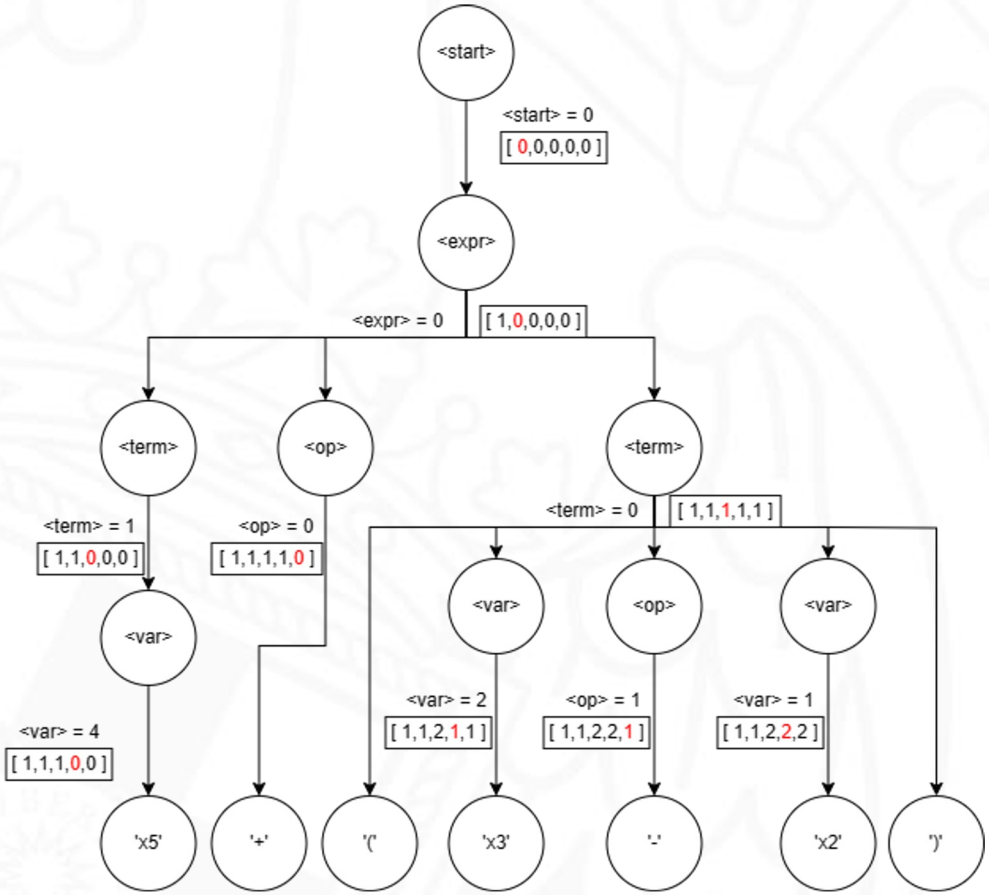
```
<start> ::= <expr> | <term>
<expr> ::= <term> <op> <term>
<term> ::=  ( <var> <op> <var> ) | <var>
<var> ::= x1 | x2 | x3 | x4 | x5
<op> ::= + | - | *
```

Individuals Genotype

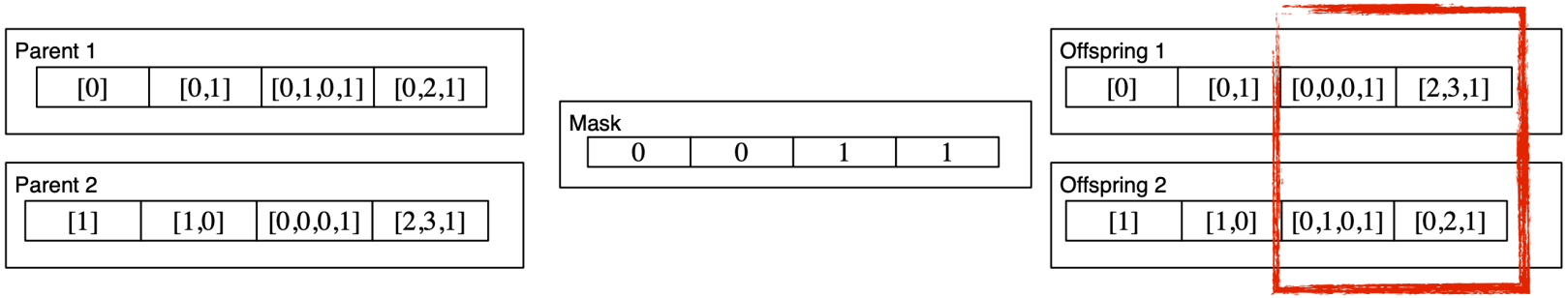
[[0] , [0] , [1,0] , [4,2,1] , [0,1]]
[<start> , <expr> , <term> , <var> , <op>]

Phenotype

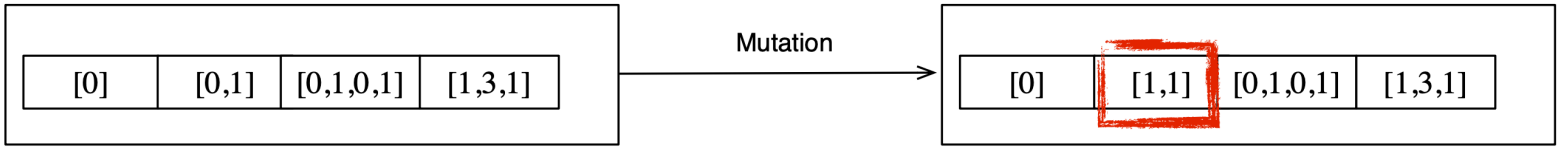
x5 + (x3 - x2)



Crossover



Mutation



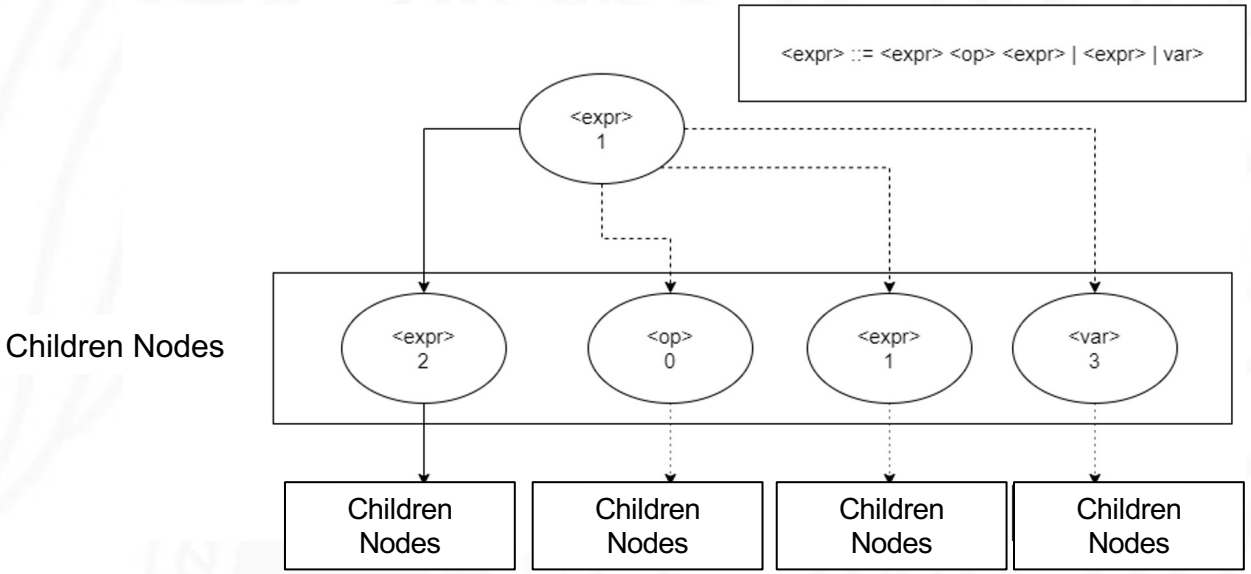
Tree-Based GE with Non-Encoding Nodes

- Tree-Based Grammatical Evolution with Non-Encoding Nodes
- T-GE-NEN combines CFG-GP and GE
- Key concept is the use of non-encoding nodes; i.e., nodes that evolve but not directly impacting the phenotype
- Tree-based representation where each node contains three attributes
  - Non-terminal symbol associated with a rule in the grammar
  - A number to select the rule's production
  - A list of potential nodes subsequent non-terminal symbols

M. De la Cruz, O. Garnica, J.M. Velasco, J.I. Hidalgo. Tree-Based Grammatical Evolution with Non-Encoding Nodes. Genetic Programming Theory and Practice XXII. Springer, 2024 (To appear)

Tree-Based GE with Non-Encoding Nodes

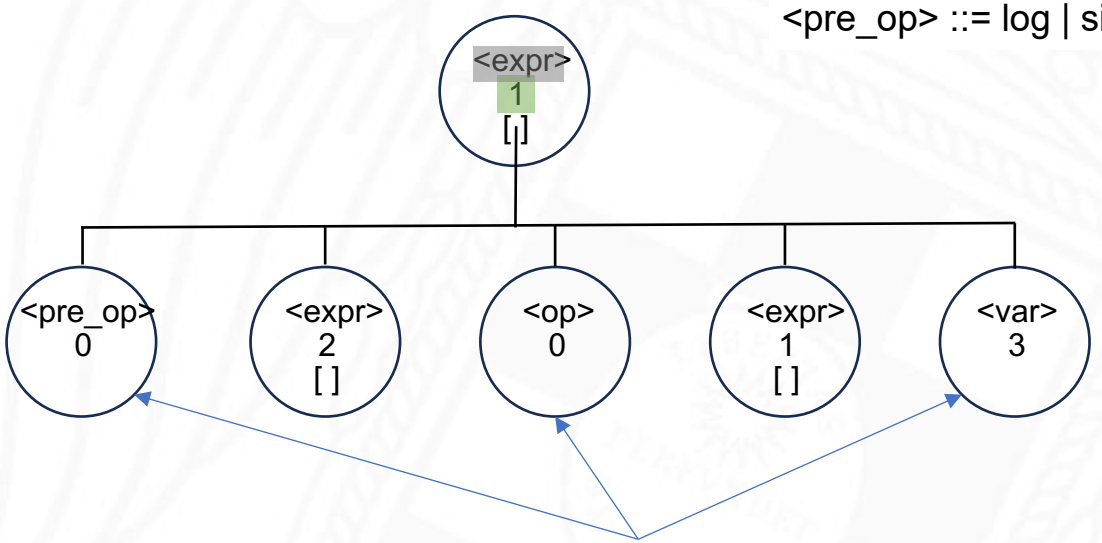
- Derivation tree with nodes associated to non-terminal types
- Dynamic generation of the child list
- Nodes:
  - Type
  - Production number
  - List of children
- Store information from previous generations



Example

- Tree-based representation where each node contains three attributes
  - Non-terminal symbol associated with a rule in the grammar
  - A number to select the rule's production
  - A list of potential nodes subsequent non-terminal symbols

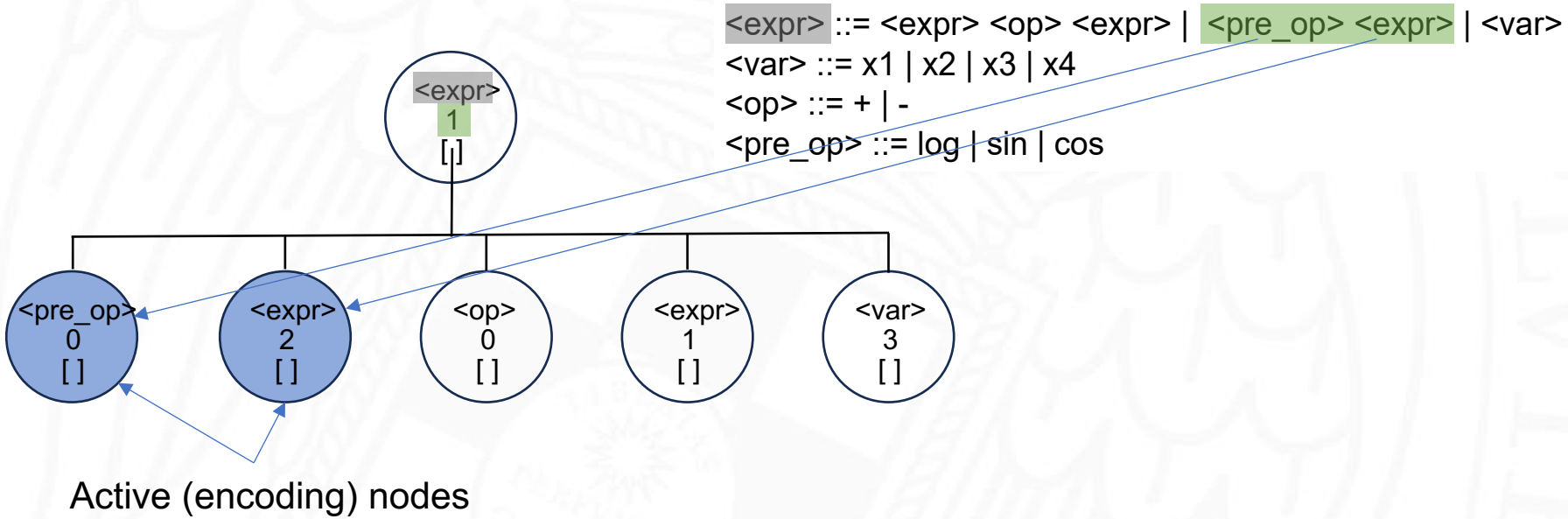
<expr> ::= <expr> <op> <expr> | <pre_op> <expr> | <var>
<var> ::= x1 | x2 | x3 | x4
<op> ::= + | -
<pre_op> ::= log | sin | cos



Do not have list of subsequent nodes. They are terminal nodes.

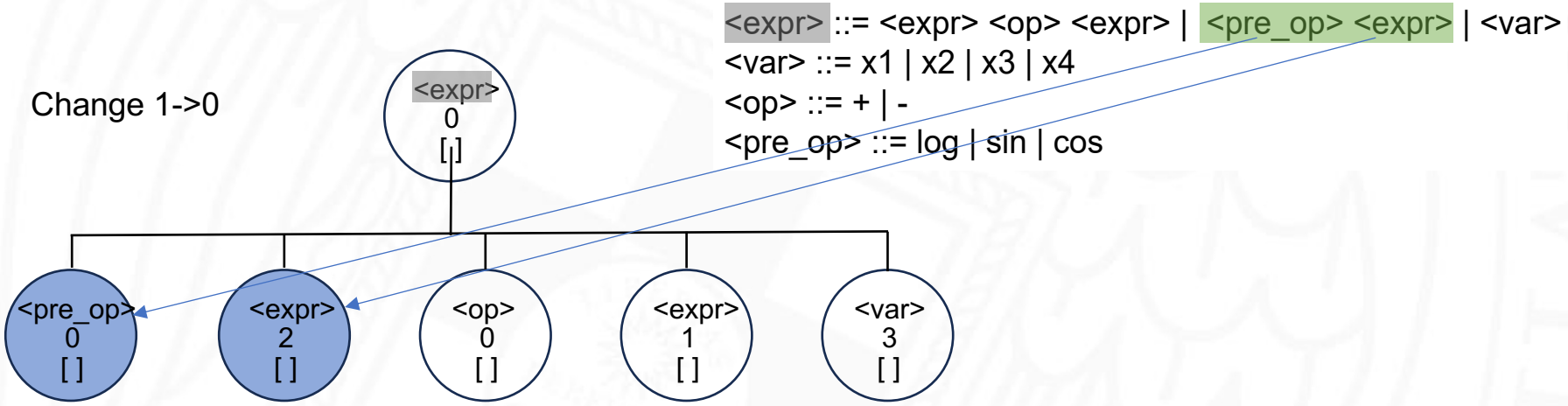
Example

- Tree-based representation where each node contains three attributes
  - Non-terminal symbol associated with a rule in the grammar
  - A number to select the rule's production
  - A list of potential nodes subsequent non-terminal symbols



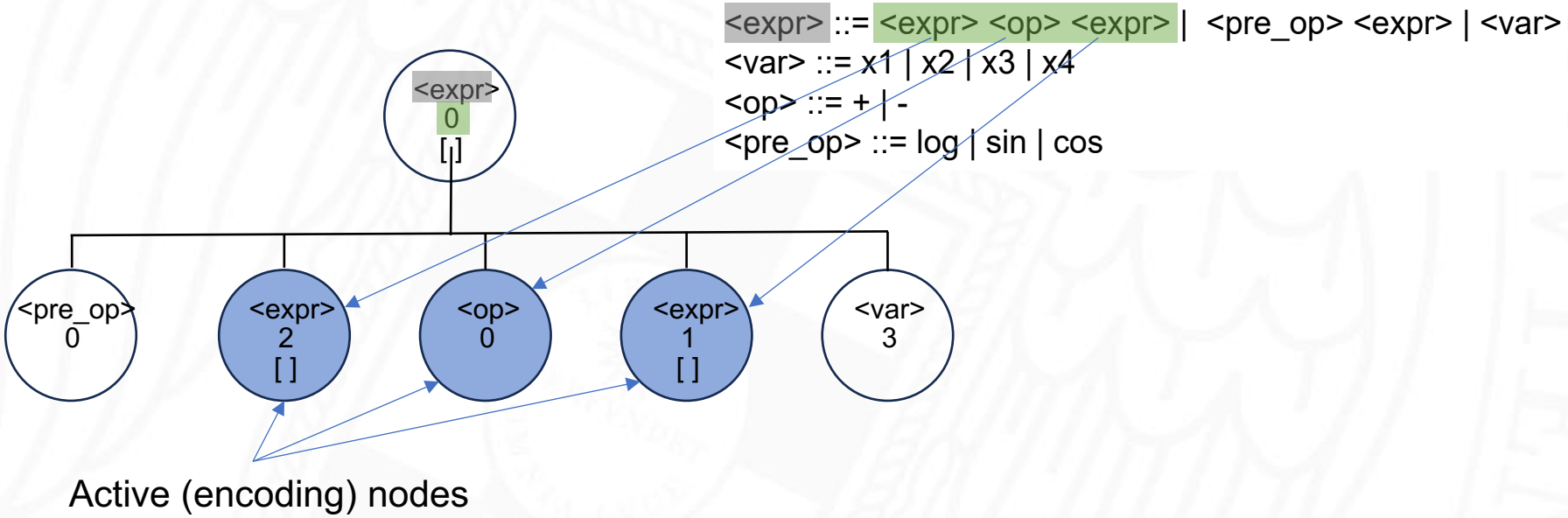
Example

- Tree-based representation where each node contains three attributes
  - Non-terminal symbol associated with a rule in the grammar
  - A number to select the rule's production
  - A list of potential nodes subsequent non-terminal symbols



Example

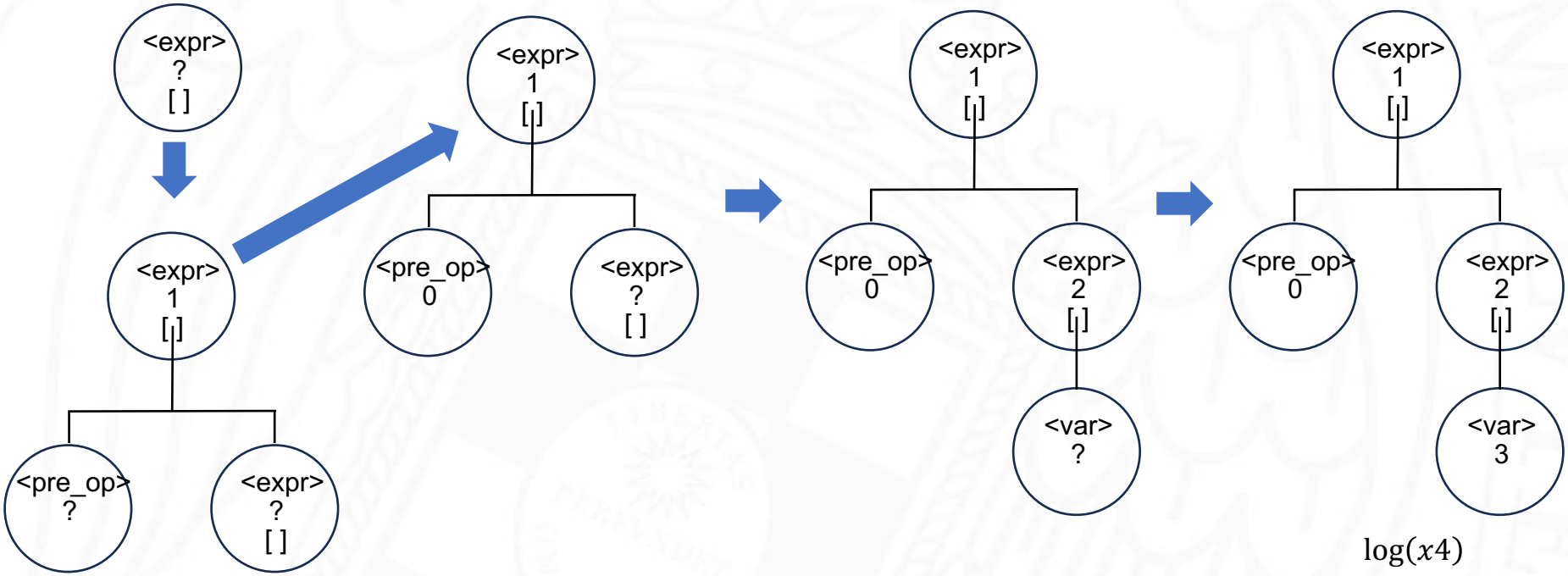
- Tree-based representation where each node contains three attributes
  - Non-terminal symbol associated with a rule in the grammar
  - A number to select the rule's production
  - A list of potential nodes subsequent non-terminal symbols



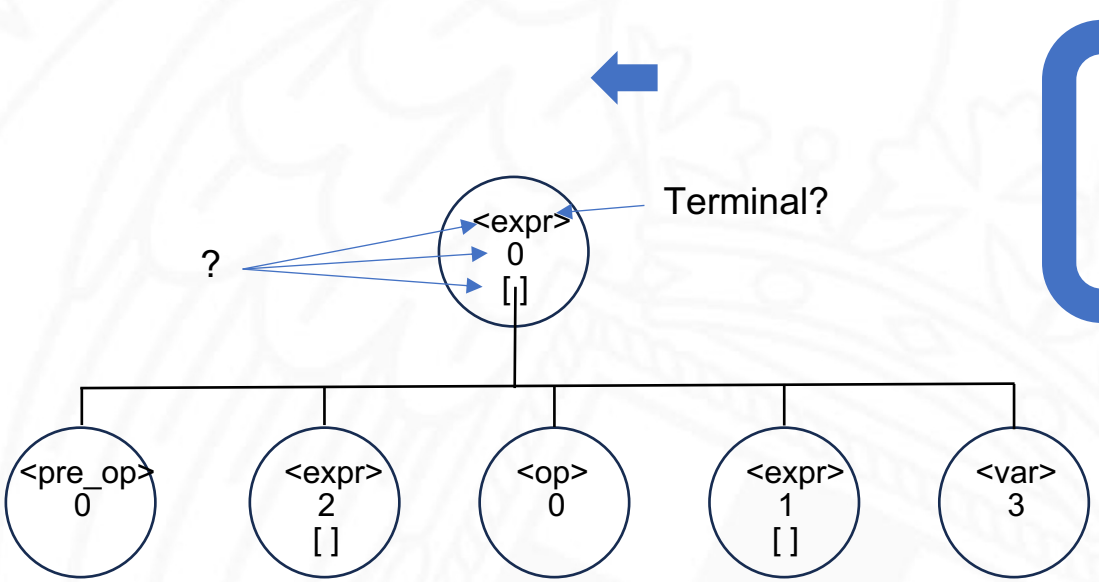
Initialization

- Resembles CFG-GP
- Only contains encoding nodes

<expr> ::= <expr> <op> <expr> | <pre_op> <expr> | <var>
<var> ::= x1 | x2 | x3 | x4
<op> ::= + | -
<pre_op> ::= log | sin | cos



Decoding



1º Traverse the tree

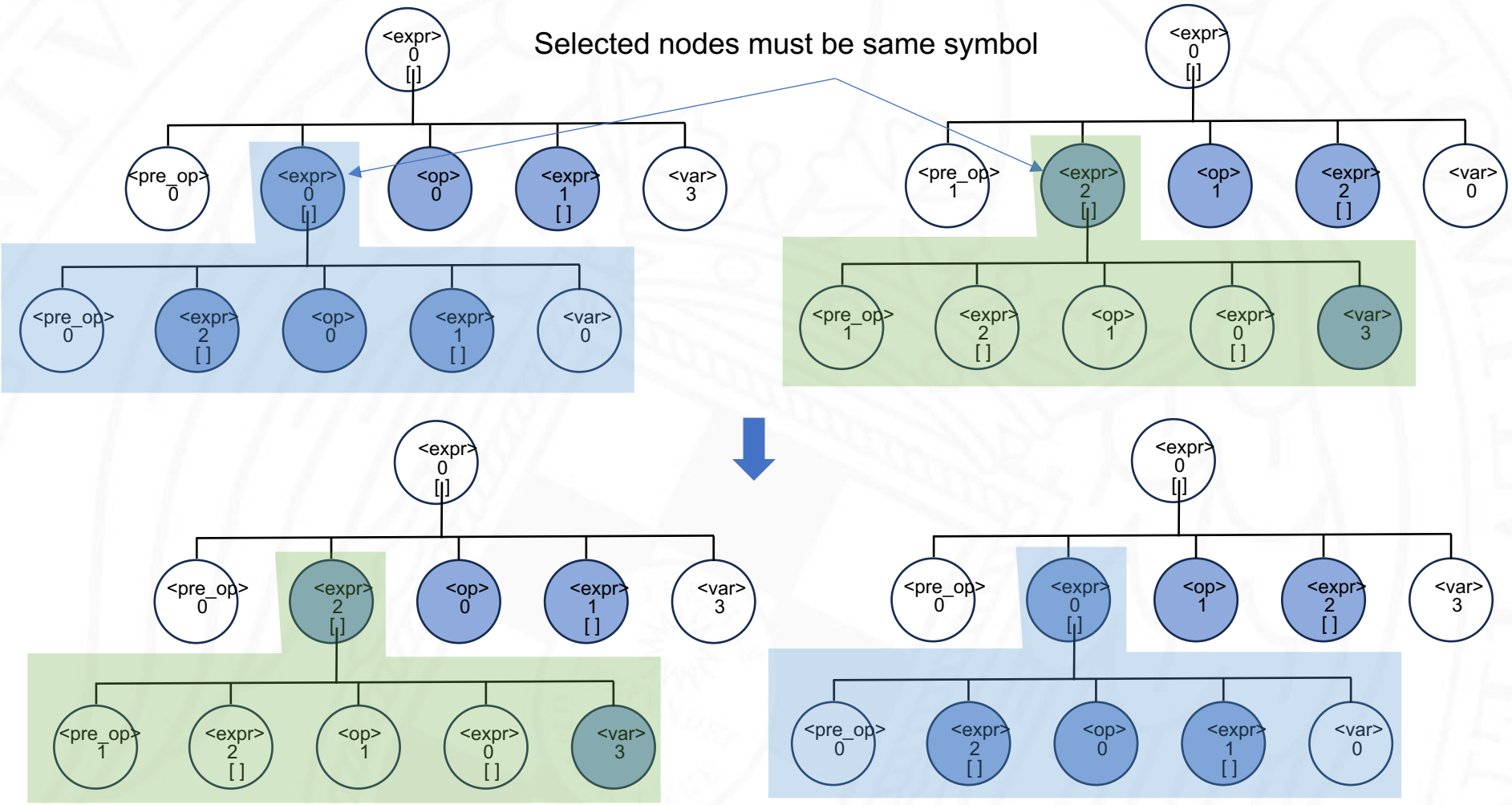
2º Validate integrity creating new nodes if needed

3º Is terminal?

4º Max Depth?

Crossover

- General sub-tree crossover with constraints

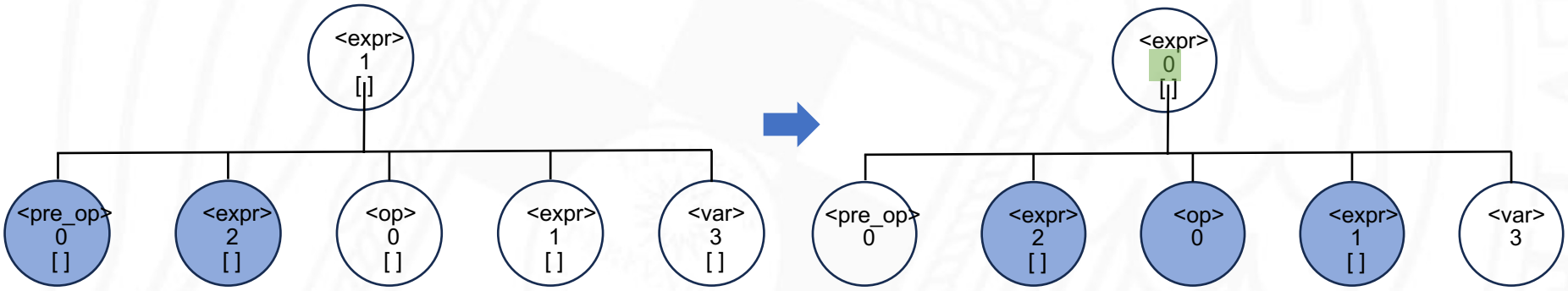


Crossover

- General sub-tree crossover with constraints
- Constraints
  - 25% times selected nodes in both trees have to be encoding
  - 50% times one of the nodes is encoding and the other is chosen randomly among available nodes (encoding and non-encoding)
  - 25% times the two nodes are chosen randomly among available nodes (encoding and non-encoding)
- These figures can be tunned for each problem
- Ensure a 75% of crossovers the resulting individual undergoes a phenotype modification

Mutation

- Targets the number in the node that identifies the production rule
- A randomly selected number between 0 and the number of productions for the non-terminal symbol
- Constraints:
  - The mutation distinguishes between encoding or encoding + non-encoding nodes
  - The selection is random (roulette Wheel) with threshold 0.75



Bloating control

- Encoding nodes
  - Max depth controlled during decoding
  - If max depth is surpassed, then substitute production rule (number) to become the shortest path to a terminal node
- Non-encoding nodes
  - Max depth controlled at the end of each generation
  - If max depth is surpassed
    - If the symbol is recursive, the recursive child nodes are deleted
    - If the production is recursive, it is modified to generate the shortest path to terminal node

GGGP vs T-GE-NEN

- Tree based GGGP
  - Tree consisting of nodes with a non-terminal type.
  - Bloating control by regeneration.
  - Mutation by branch regeneration.
  - Mutation occurs at most once
- T-GE_NEN
  - Tree consisting of nodes with a non-terminal type and a production number.
  - The list of children nodes is dynamic, with non-encoding elements.
  - Bloating control by production change.
  - Mutation can occur at each node.

Tree-Based GE with Non-Encoding Nodes

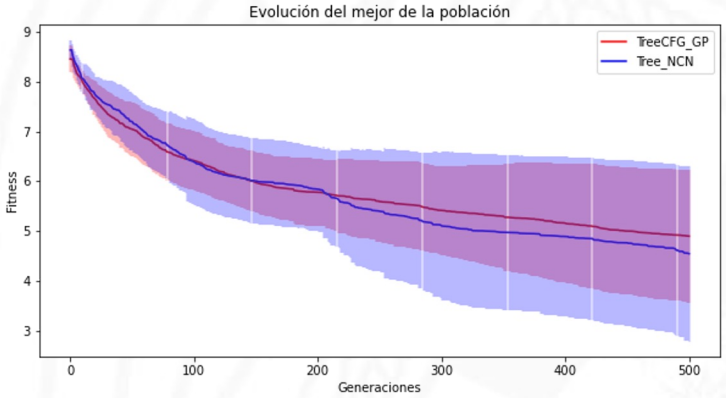
- Pros
  - T-GE-NEN offers further exploration of the search space and maintaining the evolution up to this point.
  - More significant variability than with grammar-driven trees.
- Cons
  - More memory usage
  - Control of bloating
    - Coding and non-coding genes (pruning of non-coding branches)
- Tips
  - Test the algorithm using the different parameters
  - Visualise the evolution of the population
  - Try to reduce the cost in time

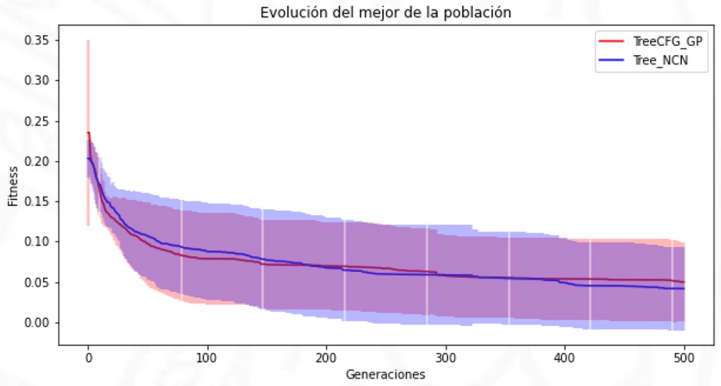
Resultados

- f(x)= (1-x)**-1

f(x)= exp(-x^2 /2) / sqrt(2*PI)

```
<start> ::= <expr>
<expr> ::= <expr><op><expr> | <number><op><expr> | (<expr>) || <pre_op>(<expr>, <number>) | <var>
<op> ::= + | * | - | /
<pre_op> ::= Math.pow
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<number> ::= <digit>.<digit><digit><digit> | <digit><digit>.<digit><digit><digit>
        | <digit><digit><digit>.<digit><digit><digit>
<var> ::= getVariable(1,k)
```

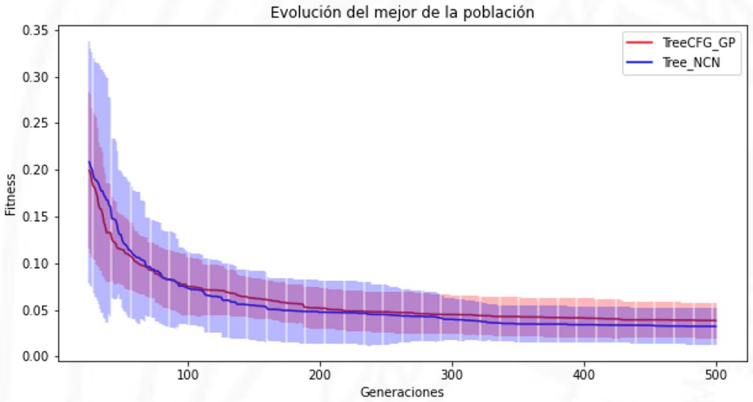




Resultados

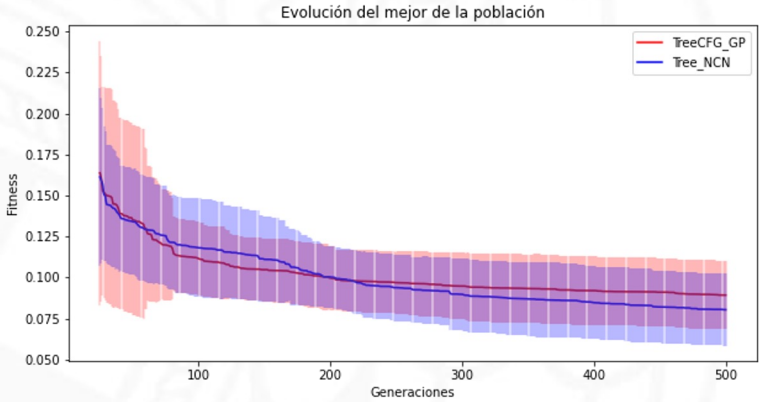
f(x) = x5 + x4 + x3 + x2 + x + 1

```
<start> ::= <expr>
<expr> ::= <expr><op><expr> | (<expr><op><expr>) | <pre_op>(<expr>) | <var>
<op> ::= + | - | * | /
<pre_op> ::= Math.sin | Math.cos| Math.exp | Math.log | inv
<var> ::= getVariable(1,k) | 1.0
```



Harmonic Curve Regression

```
<start> ::= <expr>
<expr> ::= <expr><op><expr> | (<expr>) | <pre_op>(<expr>) | <var>
<op> ::= + | *
<pre_op> ::= inv | Math.sqrt
<var> ::= getVariable(1,k)
```

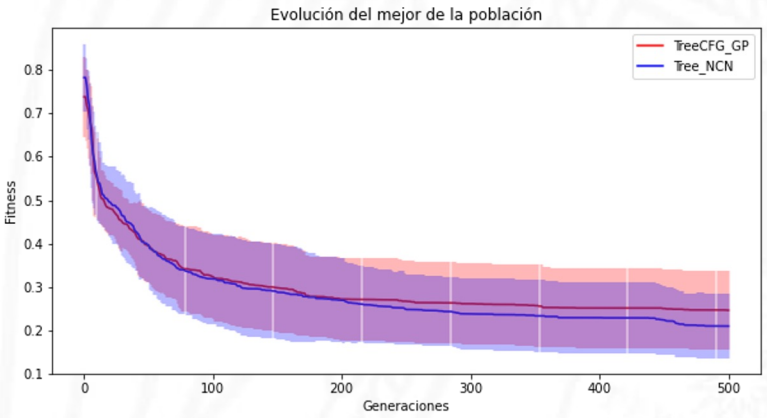


Resultados

Pagie Polynomial

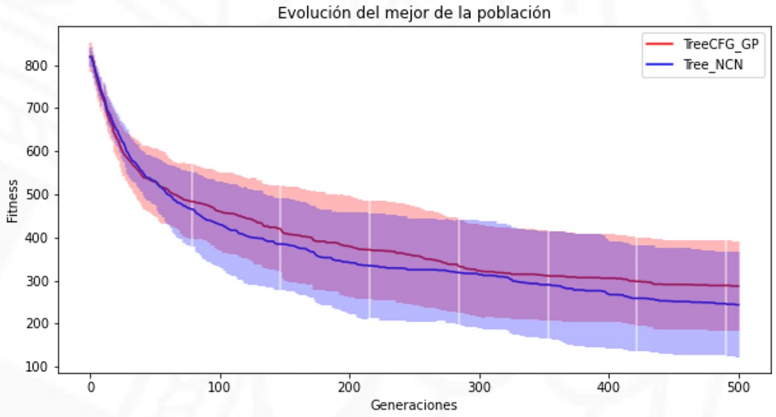
$$f(x)=1/(1+ (x^{**}(-4))) + 1/(1+(y^{**}(-4)))$$

```
<start> ::= <expr>
<expr> ::= <expr><op><expr> | (<expr>) | <pre_op>(<expr>) | <var>
<op> ::= + | - | * | /
<pre_op> ::= Math.sin | Math.cos| Math.exp | Math.log
<var> ::= getVariable(1,k) | getVariable(2,k)
```

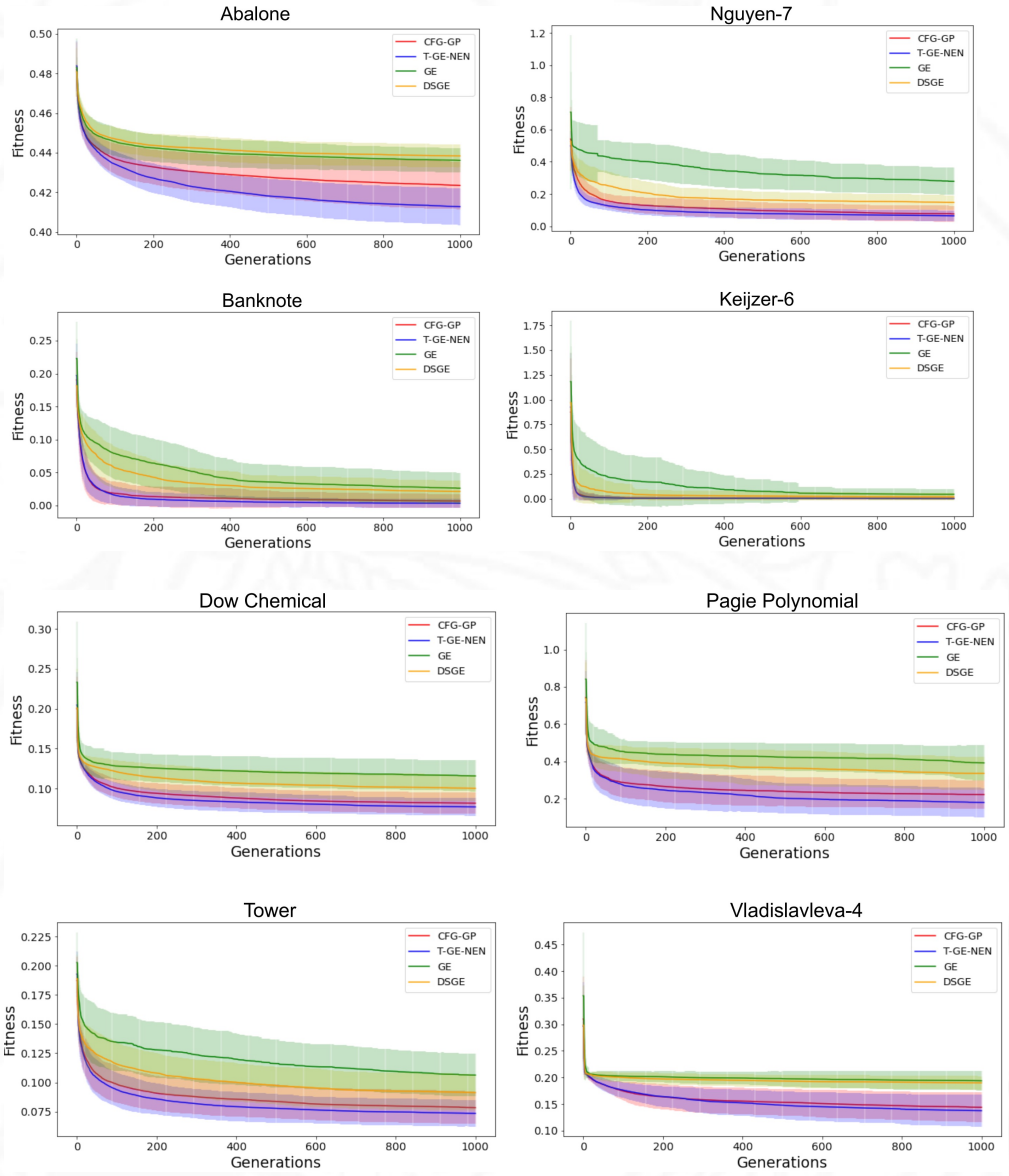


Multiplexer 8-bits

```
<start> ::= (<B>?1:0)
<B> ::= <B> "&" <B> | <B> "|" <B> | !(<B>) | !(<B> "|" <B>)
        | (<B> ? <B> : <B>) | toBool(<var>)
<var> ::= getVariable(1,k) | getVariable(2,k) | getVariable(3,k) | getVariable(4,k)
        | getVariable(5,k) | getVariable(6,k) | getVariable(7,k) | getVariable(8,k)
        | getVariable(9,k) | getVariable(10,k) | getVariable(11,k)
```



Fitness evolution



Summary

- Genetic Programming
  - Syntax trees
- Tree-based Grammar Guided GP
  - Derivation trees with crossover, mutation and initialization restricted by grammars
- Grammatical evolution
  - List of numbers decoded by grammars
- Structured Grammatical evolution (SGE & DSGE)
  - List of list of numbers
- Tree-Based GE with Non-Encoding Nodes
  - T-GE-NEN combines CFG-GP and GE
  - Non-encoding nodes

Summary

- Genetic Programming in general
  - Interpretability
    - Even explanaibility
  - Adaptability
  - Bloating!
  - Parameters!
- Grammar based GP
  - Introduce Knowledge
  - Bloating Control (xGE)
  - Grammar Design!
  - Restriction of the search space

Grammars

Regression Grammar

```
<expr>   ::= ( <expr> <op> <expr> ) | <pre_op>( <expr> ) | <term>
<op>     ::= + | - | * | /
<term>   ::= <var> | <number>
<var>    ::= x1 | ... | xn
<number> ::= (- <digit>.<digit><digit><digit>  )
           | <digit>.<digit><digit><digit>
<pre_op> ::= sin | cos| exp | log | inv | sqrt | tan
<digit>  ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
```

Classification Grammar

```
<type>    ::= if( <binexpr> )then class_1 elseif( <binexpr> ) then class_2
              ... else class_n
<binexpr> ::= ( <expr> <relop> <expr> )
            | ( ( <expr> <relop> <expr> ) <binop> <binexpr> )
<binop>   ::= && | ||
<expr>    ::= ( <expr> <op> <expr> ) | <pre_op>( <expr> ) | <term>
<op>      ::= + | - | * | /
<term>    ::= <var> | <number>
<var>     ::= x1 | ... | xn
<number>  ::= (- <digit>.<digit><digit><digit>  ) |
               <digit>.<digit><digit> <digit>
<pre_op>  ::= sin | cos| exp | log | inv | sqrt | tan
<digit>   ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<relop>   ::= < | > | <= | >=
```

Grammar design

- Include knowledge into the grammar
  - Poli10: X1*X2+X3*X4+X5*X6+X1*X7*X9+X3*X6*X10

poli10.bnf — Modificado

```
<func> ::= <expr>

<expr> ::= <preop> (<expr>) | <var> | <expr> <op> <expr> | <expr> <op> <expr>

<op> ::= +|-|/|*

<var> ::= getVariable(1,k)|getVariable(2,k)|getVariable(3,k)|getVariable(4,k)|
getVariable(5,k)|getVariable(6,k)|getVariable(7,k)|getVariable(8,k)|
getVariable(9,k)|getVariable(10,k)

<preop>::= -|Math.sin|Math.cos
```

Grammar design

```
<func> ::= <expr>
<expr> ::= <expr> <op> <expr>|<expr> <op> <expr>|<expr> <op> <expr> <op> <expr> | <var>
<op> ::= +|*
<var> ::= getVariable(1,k)|getVariable(2,k)|getVariable(3,k)|getVariable(4,k)
|getVariable(5,k)|getVariable(6,k)|getVariable(7,k)|getVariable(8,k)
|getVariable(9,k)|getVariable(10,k)
```

trickypoli10.bnf

```
<func> ::= <expr>
<expr> ::= <var> <op> <var> + <var> <op> <var> + <var> <op> <var>
          + <var> <op> <var> <op> <var> + <var> <op> <var> <op> <var>
<op> ::= +|*
<var> ::= getVariable(1,k)|getVariable(2,k)|getVariable(3,k)|getVariable(4,k)
|getVariable(5,k)|getVariable(6,k)|getVariable(7,k)|getVariable(8,k)
|getVariable(9,k)|getVariable(10,k)
```

Grammar and Fitness function

```
<type> ::= if( <binexpr> ){result= 0}else{result= 1}
<binexpr> ::= ( <expr> <relop> <mix> ) | ( ( <expr> <relop> <mix> ) <binop> <binexpr> )
<binop> ::= && | ||
<expr> ::= <term> | <term> <op> <expr> | ( <term> <op> <expr> ) | ( <number> <op> <expr> )
        |( <expr> <op> <number> )  | <number> <op> <expr> | <expr> <op> <number>
        | ( <expr> <op> <expr> ) | <expr> <op> <expr>
<op> ::= + | - | * | /
<term> ::= <gluc> | <HR> | <steps> | <Cal>
<gluc> ::= gluc(t) | gluc(t-5) | gluc(t-10) | gluc(t-15) | gluc(t-20) | gluc(t-25)
        | gluc(t-50) | gluc(t-75) | gluc(t-100)
<HR> ::= hr(t) |  hr(t-20) |  hr(t-40) |  hr(t-60) |  hr(t-80) | hr(t-100) |  hr(t-115)

<steps> ::= steps(t) | steps(t-15) | steps(t-30) | steps(t-45) | steps(t-60) | steps(t-75)
        | steps(t-90)| steps(t-105) | steps(t-120)
<Cal> ::= cal(t) | cal(t-20) | cal(t-40) | cal(t-60) | cal(t-80) | cal(t-100) | cal(t-115)
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<mix> ::= <number> | <expr>
<number> ::= <digit>.<digit><digit><digit> | <digit><digit>.<digit><digit><digit>
        | <digit><digit><digit>.<digit><digit><digit>
<relop> ::= < | > | <= | >=
```

$$\mathrm{WA} = 0.5 * Accuracy + 0.5 * F1_{measure}$$

$$F1_{measure} = \frac{2 * Recall * Precision}{Recall + Precision}$$

$$Recall = \frac{TruePositives}{TotalPositives}$$

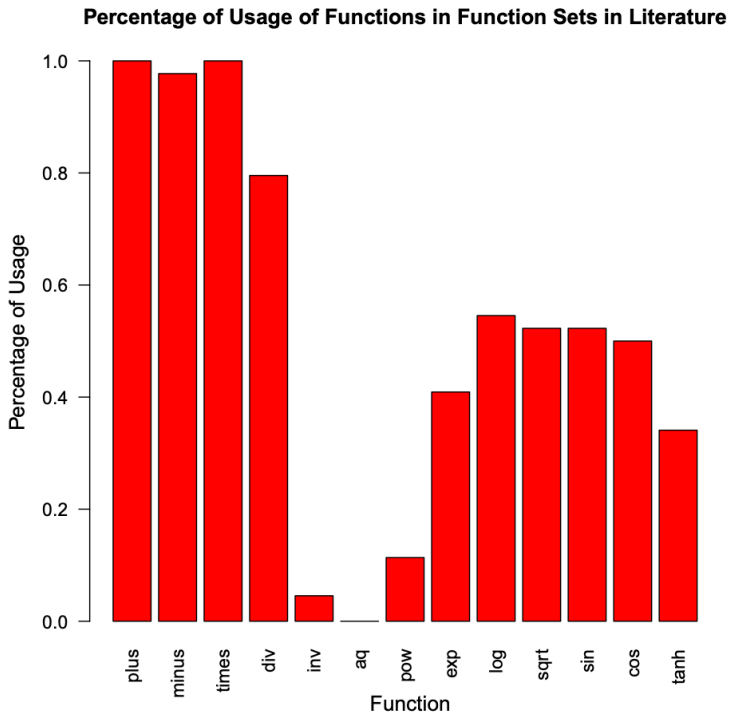
Grammar design

- Grammar is a key input to GE
  - The performance of GE is significantly influenced by the design and structure of the grammar
    - Search space accesability
  - Grammar definition is generally performed by the users of GE, solutions developers or domain experts, each of whom hand-craft the grammar,
  - Often using the same grammar from similar problems.
  - The choice of terminals and non-terminals, and their composition to form production rules, is largely based on expertise.
  - The choice of the can have a vital impact on the performance of GP.

Nicolau M, Agapitos A. Understanding grammatical evolution: grammar design. In: Ryan C, O'Neill M, Collins JJ, editors. *Handbook of grammatical evolution, Chap. 2.* Cham, Switzerland: Springer; 2018. pp. 23–53.

Grammar Design

- Grammar design



Regression Grammar

```
<expr>   ::= ( <expr> <op> <expr> ) | <pre_op>( <expr> ) | <term>
<op>     ::= + | - | * | /
<term>   ::= <var> | <number>
<var>    ::= x1 | ... | xn
<number> ::= (- <digit>.<digit><digit><digit>  )
           | <digit>.<digit><digit><digit>
<pre_op> ::= sin | cos| exp | log | inv | sqrt | tan
<digit>  ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
```

Nicolau M, Agapitos A. Understanding grammatical evolution: grammar design. In: Ryan C, O'Neill M, Collins JJ, editors. *Handbook of grammatical evolution, Chap. 2.* Cham, Switzerland: Springer; 2018. pp. 23–53.

Grammar Design

Avoid to have more recursive productions than non-recursive

```
<s> ::= <e>
<e> ::= <e> <o> <e>
      | ( <e> <o> <e> )
      | <f> ( <e> , <e> )
      | <v>
<o> ::= + | - | *
<f> ::= pdiv
<v> ::= x | 1.0
```

```
<s> ::= <e>
<e> ::= <e> <o> <e> | <v>
      | ( <e> <o> <e> ) | <v>
      | <f> ( <e> , <e> ) | <v>
<o> ::= + | - | *
<f> ::= pdiv
<v> ::= x | 1.0
```

Grammar 1 Balanced recursion grammar

```
<s> ::= <e>
<e> ::= <e> <o> <e> | <v>
      | ( <e> <o> <e> ) | <v>
      | <f> ( <e> , <e> ) | <v>
<o> ::= + | + | + | + | + | +
      | - | - | - | - | - | -
      | * | * | * | * | * | *
<f> ::= pdiv
<v> ::= x | x | x | x | x | x
      | x | x | x | x | x | x
      | x | x | x | x | x | x
      | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0
      | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0
      | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0
```

Grammar 2 Unlinked productions grammar

Nicolau M, Agapitos A. Understanding grammatical evolution: grammar design. In: Ryan C, O'Neill M, Collins JJ, editors. *Handbook of grammatical evolution, Chap. 2.* Cham, Switzerland: Springer; 2018. pp. 23–53.

Grammar Design

```
<e> ::= <e> + <e> | <e> - <e> | <e> * <e>
      | <e> + <e> | <e> - <e> | <e> * <e>
      | x | x | x
      | 1.0 | 1.0 | 1.0
      | ( <e> + <e> ) | ( <e> - <e> ) | ( <e> * <e> )
      | ( <e> + <e> ) | ( <e> - <e> ) | ( <e> * <e> )
      | x | x | x
      | 1.0 | 1.0 | 1.0
      | pdiv ( <e> , <e> ) | pdiv ( <e> , <e> )
      | pdiv ( <e> , <e> )
      | pdiv ( <e> , <e> ) | pdiv ( <e> , <e> )
      | pdiv ( <e> , <e> )
      | x | x | x
      | 1.0 | 1.0 | 1.0
```

Nicolau M, Agapitos A. Understanding grammatical evolution: grammar design. In: Ryan C, O'Neill M, Collins JJ, editors. *Handbook of grammatical evolution, Chap. 2.* Cham, Switzerland: Springer; 2018. pp. 23–53.

Grammar Design

```
<e> ::= ( <e> + <e> )
      | <e> + <e>
      | ( <e> - <e> )
      | <e> - <e>
      | ( <e> * <e> )
      | <e> * <e>
      | <e> pdiv <e>
      | ( <e> pdiv <e> )
      | x | x | x | x
      | 1.0 | 1.0 | 1.0 | 1.0
```

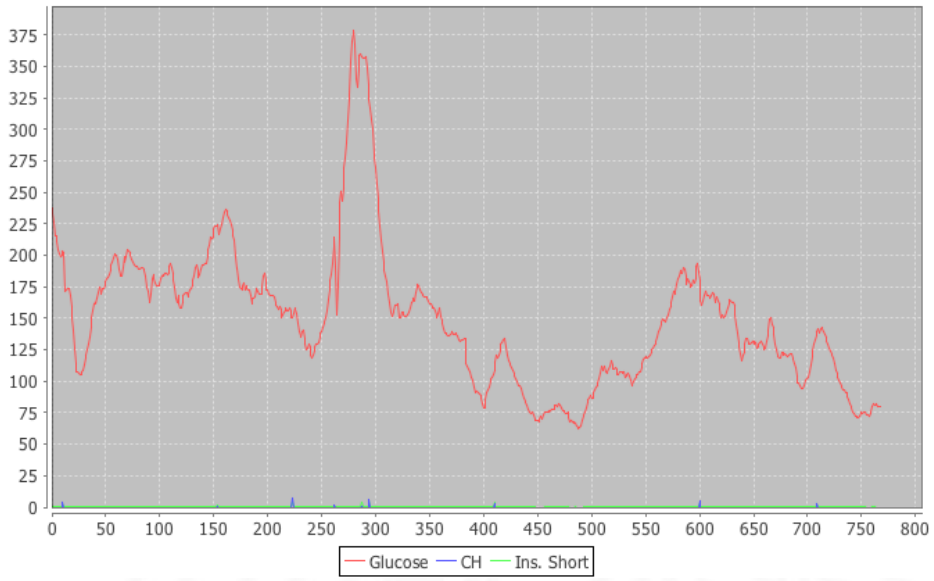
Grammar 4 Corrected-biases grammar

Nicolau M, Agapitos A. Understanding grammatical evolution: grammar design. In: Ryan C, O'Neill M, Collins JJ, editors. *Handbook of grammatical evolution, Chap. 2.* Cham, Switzerland: Springer; 2018. pp. 23–53.

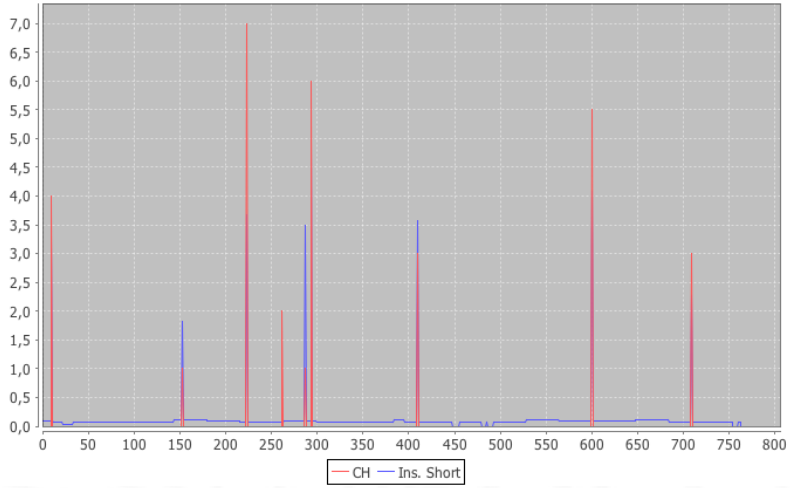
GE Modeling Example

Glucose Levels in People with diabetes

Model and prediction



- CH spikes are meals.
- INS spikes are bolus.



- What´s glucose level I will have in **m** minutes if :
  - I eat **X** carbohydrates
  - I inject **Y** units of insuline Type **A**
  - Historical Data
- In other words
  - Know the ammount of insuline neccesary to have a good glycemic control

Model Extraction with GE

- The grammar allows the inclusion of problem knowledge.
- GE uses integer chromosomes.
- Allows typical GA crossover and mutation operators.

```
# Model expression
<func> ::= <exprgluc> + <exprch> - <exprins>

# Glucose
<exprgluc> ::= (<exprgluc> <op> <exprgluc>) | <preop>
      (<exprgluc>) | (<cte> <op> <exprgluc>)
         | predictedData(t-<idx
         >) | realData(t-<idx2hOrMore>)

# CH
<exprch> ::= (<exprch> <op> <exprch>)
         |<preop> (<exprch>)
         |(<cte> <op> <exprch>)
         |(getPrevData(1,t,1) * <cte> * <curvedCH>)

# Insulin:
## Sum of insulins in past 2h minus the peak
## Curve for the peak in past 2h
<exprins> ::= (<exprins> <op> <exprins>)
         |<preop> (<exprins>)
         |(<cte> <op> <exprins>)
         |getVariable(2,t-<idx>)
```

BNF Grammars for Glucose Modelling

```
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> + <exprch> - <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
```

Model Extraction with Grammatical Evolution

- Separate the data into time slots.
- Generate different models for each principal meal of the day.
- Problem in the frontiers between time slots.

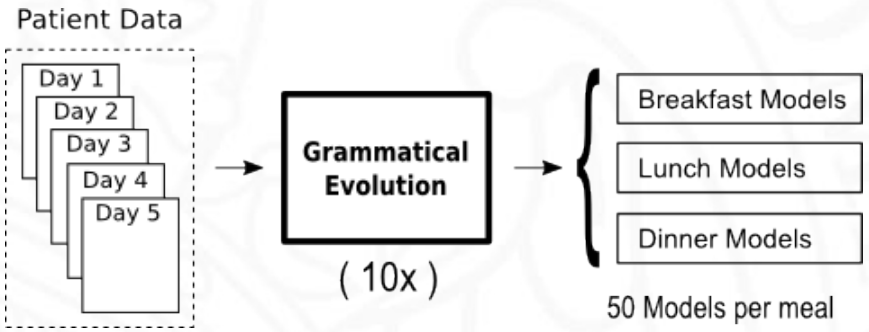


Figure 6: Training process in the GE approach for one patient.

Adaptability

- t: Time of prediction
  - Glucose values:
    `Glucose(t), Glucose(t-5), Glucose(t-10)... Glucose(t-120)`
  - Heart Rate values:
    `HR(t), HR(t-5), HR(t-10)... HR(t-120)`
  - Amount of steps:
    `Steps (t), Steps (t-5), Steps (t-10)... Steps (t-120)`
  - Burned calories:
    `Calories (t), Calories(t-5), Calories(t-10)... Calories(t-120)`

Prediction for k ∈ [30,60,90,120] minutes.

- Class of prediction:
  - Hypoglycemia = 0
  - Non-hypoglycemia = 1

Example model structure:

```
if(((837.055/944.323+ Calories(t-20) -( HR(t-100) - Glucose(t) ))   <=33.128-(Glucose(t-
75)/HR(t-60))+ Calories(t-80) + Steps(t-90) /735.526))

      {result=0}

  else {result=1}
```

Conclusions

- We have presented different techniques to obtain customized glucose models for diabetic patients.
- We have worked with data from real patients from Spanish Hospitals.
- Our results are diverse:
  - Genetic Programming and k-Nearest-Neighbor obtain good short-term predictions.
  - Grammatical Evolution models can be good when focusing on meal times.
  - Preprocessing is important
    - Latent variables
    - Clustering
  - Data Variation helps to reduce number of dangerous predictions

Advanced Topics -Lexicase selection

- Parent selection technique based on lexicographic ordering of test (i.e. fitness) cases.
- Each parent selection event proceeds as follows:
  - 1. The entire population is added to the selection pool.
  - 2. The fitness cases are shuffled.
  - 3. Individuals in the pool with a fitness worse than the best fitness on this case among the pool are removed.
  - 4. If more than one individual remains in the pool, the first case is removed and 3 is repeated with the next case.
    - If only one individual remains, it is the chosen parent.
    - If no more fitness cases are left, a parent is chosen randomly from the remaining individuals.
- Epsilon-lexicase selection for regression problems

Lexicase selection

Lexicase Selection Algorithm:

To Pick One Parent

1. pool ← population
2. cases ← list of training cases, shuffled
3. while |pool| > 1 and |cases| > 0:
   a. t ← first case in cases
   b. best ← the best error value of any individual in pool on case t
   c. pool ← filter pool to include only individuals with error of best on t
   d. pop t from cases
4. if |pool| = 1:
   a. return the one individual in pool
5. else:
   a. return random individual from pool

Thomas Helmuth and William La Cava. 2022. Lexicase selection. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1385–1397. https://doi.org/10.1145/3520304.3533633

Lexicase selection

- Minimize error
- Case order 5,2,1,3,4

Case	X	X	X	X	15	X
3	5	8	0	14	0	2
4	15	8	0	15	100	16
Total Error	45	38	134	57	128	49

Thomas Helmuth and William La Cava. 2022. Lexicase selection. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1385–1397. https://doi.org/10.1145/3520304.3533633

Lexicase selection

Lexicase Selection: Example 1

Case order: 5, 2, 1, 3, 4

- 5: best is 1, pool = {C, D, E}
- 2: best is 12, pool = {D, E}
  - *Note: best is always relative to pool, not full population*
- 1: best is 15, pool = {D, E}
- 3: best is 0, pool = {E}
- return E

	Individual				
Case	~~A~~	~~B~~	~~C~~	~~D~~	E
1	10	8	73	15	15
2	5	7	60	12	12
3	5	8	0	14	0
4	15	8	0	15	106
5	10	7	1	1	1
Total Error:	**45**	**38**	**134**	**57**	**134**

Thomas Helmuth and William La Cava. 2022. Lexicase selection. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1385–1397. https://doi.org/10.1145/3520304.3533633

Lexicase selection

Lexicase Selection: Example 2

Case order: 1, 2, 5, 4, 3

- 1: best is 8, pool = {B}
- return B

	Individual				
Case	~~A~~	B	~~C~~	~~D~~	~~E~~
1	10	8	73	15	15
2	5	7	60	12	12
3	5	8	0	14	0
4	15	8	0	15	106
5	10	7	1	1	1
Total Error:	**45**	**38**	**134**	**57**	**134**

Thomas Helmuth and William La Cava. 2022. Lexicase selection. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1385–1397. https://doi.org/10.1145/3520304.3533633

Lexicase selection

Lexicase Selection: Example 3

Case order: 3, 5, 4, 1, 2

- 3: best is 0, pool = {C, E}
- 5: best is 1, pool = {C, E}
- 4: best is 0, pool = {C}
- return C

	Individual				
Case	~~A~~	~~B~~	C	~~D~~	~~E~~
1	10	8	73	15	15
2	5	7	60	12	12
3	5	8	0	14	0
4	15	8	0	15	106
5	10	7	1	1	1
Total Error:	**45**	**38**	**134**	**57**	**134**

Thomas Helmuth and William La Cava. 2022. Lexicase selection. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1385–1397. https://doi.org/10.1145/3520304.3533633

Lexicase selection

- When it's applicable
  - When fitness can be decomposed into component parts.
    - Ex: summations / averages over cases (mean squared error, etc) –
- Places it doesn't apply:
  - Single output, black-box function optimization
- How many fitness components?
  - There are factorial(n) different shufflings of n cases
  - Lexicase can select from at most that number of different error vectors
  - 4! = 24 isn't much if you have a population such as 1000
  - 6! = 720 is often reasonable

Thomas Helmuth and William La Cava. 2022. Lexicase selection. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1385–1397. https://doi.org/10.1145/3520304.3533633

Epsilon-Lexicase selection

epsilon-Lexicase Selection Algorithm:

To Pick One Parent

```
1. pool ← population
2. cases ← list of training cases, shuffled
3. while |pool| > 1 and |cases| > 0:
   a. t ← first case in cases
   b. best ← the best error value of any individual in pool on case t
   c. epsilon ← median absolute deviation of population on case t
   d. pool ← filter pool to include only individuals within epsilon of
      best
   e. pop t from cases
4. if |pool| = 1:
   a. return the one individual in pool
5. else:
   a. return random individual from pool
```

Thomas Helmuth and William La Cava. 2022. Lexicase selection. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1385–1397. https://doi.org/10.1145/3520304.3533633

Advanced Topics-Probabilistic GE

- Probabilistic Context-Free Grammar (PCFG) where its probabilities are adapted during the evolutionary process, taking into account the productions chosen to construct the fittest individual.
- The genotype is a list of real values, where each value represents the likelihood of selecting a derivation rule.
- A PCFG is a quintuple P G = (NT, T, S, P, Probs)
  - set of Non-Terminal (NT)
  - Set of Terminal (T)
  - S is an element of NT called the axiom,
  - P is the set of production rules
  - Probs is a set of probabilities associated with each production rule.

Advanced Topics-Probabilistic GE

<start>	::= <expr>	(1.0)
<expr>	::= <expr><op><expr>	(0.5)
	| <var>	(0.5)
<op>	::= +	(0.33)
	| *	(0.33)
	| −	(0.33)
<var>	::= x	(0.5)
	| 1.0	(0.5)

Genotype

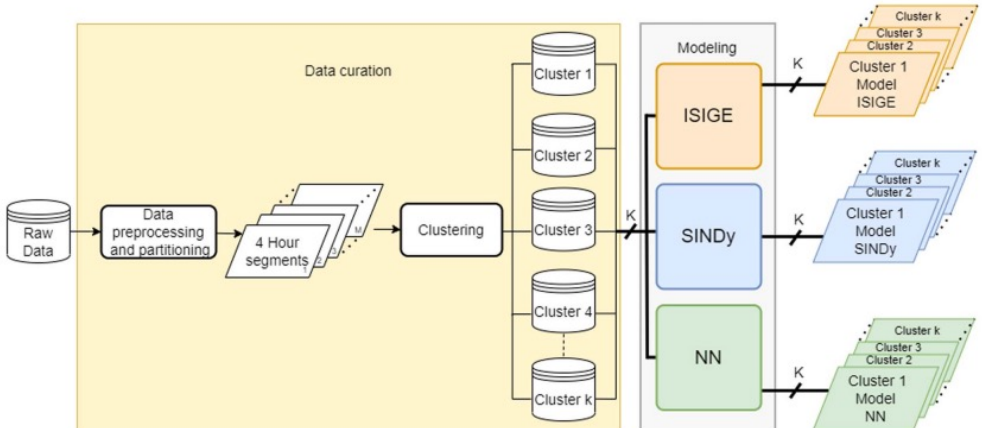
[0.8, 0.2, 0.98, 0.45, 0.62, 0.37, 0.19]

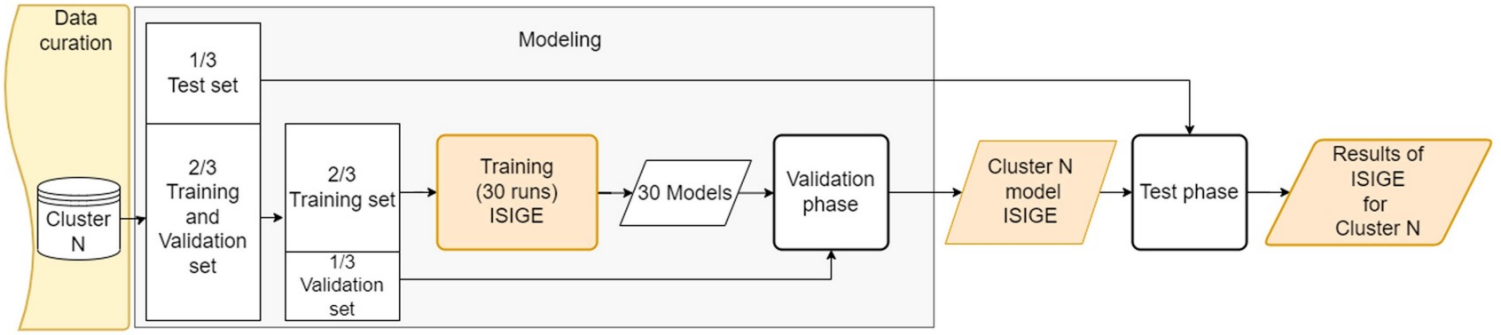
<start>	→ <expr>	(0.8)
<expr>	→ <expr><op><expr>	(0.2)
<expr><op><expr>	→ <var><op><expr>	(0.98)
<var><op><expr>	→ x <op><expr>	(0.45)
x<op><expr>	→ x * <expr>	(0.62)
x * <expr>	→ x * <var>	(0.37)
x * <var>	→ x * x	(0.19)

Mégane, J., Lourenço, N., & Machado, P. (2022, July). Probabilistic structured grammatical evolution. In *2022 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1-9). IEEE.

Advanced Topics - ISIGE

- Interpretable Sparse Identification by Grammatical Evolution (ISIGE)





D. Parra, D. Joedicke, J. M. Velasco, G. Kronberger and J. I. Hidalgo, "Learning Difference Equations With Structured Grammatical Evolution for Postprandial Glycaemia Prediction," in *IEEE Journal of Biomedical and Health Informatics*, vol. 28, no. 5, pp. 3067-3078, May 2024, doi: 10.1109/JBHI.2024.3371108.

Software

- JECO: Java Evolutionary COmputation library.
  - GE and more EA
  - https://github.com/ABSysGroup/jeco
- PonyGE2
  - https://github.com/PonyGE/PonyGE2/
- Structured GE
  - https://github.com/nunolourenco/sge3
- Genetic Engine
  - A hybrid between strongly-typed (STGP) and grammar-guided genetic programming (GGGP).
  - https://pypi.org/project/GeneticEngine/

References

- McKay, R.I., Hoai, N.X., Whigham, P.A. et al. Grammar-based Genetic Programming: a survey. Genet Program Evolvable Mach 11, 365–396 (2010). https://doi.org/10.1007/s10710-010-9109-y
- Kronberger, G., Colmenar, J.M., Winkler, S.M. Hidalgo, J.I. Multilayer analysis of population diversity in grammatical evolution for symbolic regression. Soft Comput 24, 11283–11295 (2020). https://doi.org/10.1007/s00500-020-05062-9

Thanks

- Several Slides and graphics produced by
  - Oscar Garnica
  - Marina de la Cruz
  - Jose Manuel Colmenar
  - J. Manuel Velasco
  - Bill La Cava
  - Thomas Helmunth
  - Jessica Megane
  - Nuno Lourenço

Acknowledgement

Work supported the Spanish Ministry of Economy and Competitiveness grants number PID2021-125549OB-I00 and PDC2022-133429-I00 co-financed by the EU Next Generation and by the Comunidad de Madrid grant IND2020/TIC-17435 co-financed by European Union FEDER Funds.

Fundación Eugenio Rodriguez Pascual
Ayudas y Premios a la Investigación en Ciencias Médicas

GOBIERNO DE ESPAÑA
MINISTERIO DE ECONOMÍA, INDUSTRIA Y COMPETITIVIDAD

UNIVERSIDAD COMPLUTENSE MADRID

GenObIA CM

Comunidad de Madrid

FEDER
Fondo Europeo de Desarrollo Regional