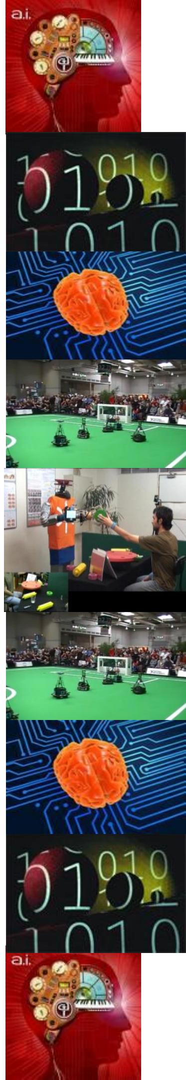
Grammatical Evolution

Context Free Grammar

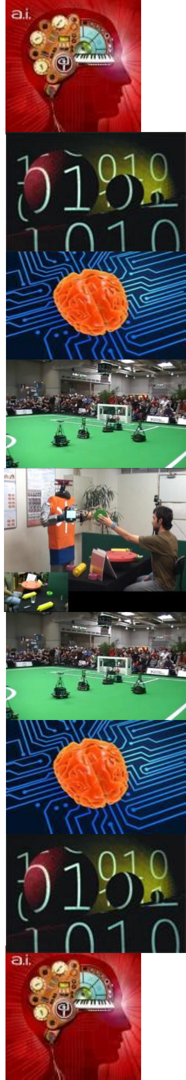
- A context-free grammar (CFG) is a formal grammar which consists of production rules which are used to map possible patterns.
- The notation used to describe such a form is called the Backus Naur Form (BNF)
- It’s a programming language grammar



Context Free Grammar

- A grammar **G = < N, T, P, S >**

- N = non-terminals denoted by **< >** which means whatever is inside can be simplified or replaced by some other attributes.
- T - terminal final values.
- P- Production rules
- S - starting value



Context Free Grammar Example

Rule_num: 0, 1, 2, 3, 4, 5, 6

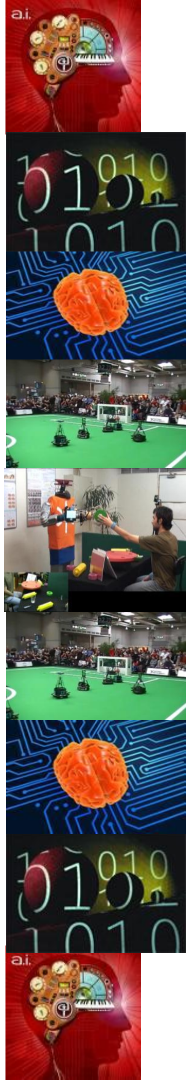
<int> –> 1|2|3|4|5|6|7

LHS RHS

N(non) T (terminal)

N can be replaced by a 1 |(OR) 2 |(OR) 3 |(OR) ....

1 production rule 0, 2 production rule 1, 3 production rule 2 etc.



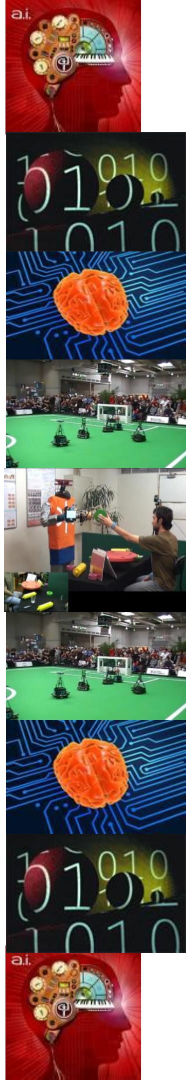
Context Free Grammar

<int> → 1|2|3|4|5|6|7

<nums> → <int> |<int> <nums>

—--

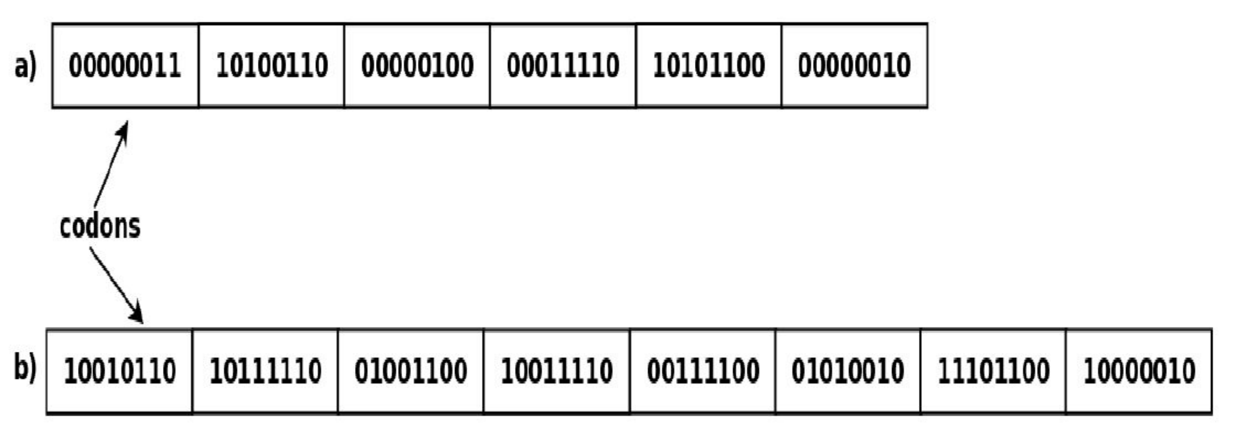
1. S-> **<nums> = <int><nums>** (left most derivation of rhs)
2. <nums> = 4 <nums> // <int> replaced by 4 rule 3
3. <nums> = 4 <int><nums> // <nums> replaced rule 1
4. <nums> = 4 1 <nums> // <int> replaced rule 0
5. <nums> = 4 1 <int> // <nums> replaced by rule 0
6. <nums> = 4 1 6 // <int> replaced by rule 5

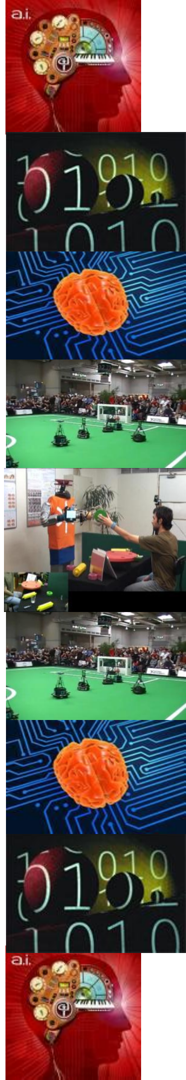


Grammatical Evolution

Like GP searches in the program space

- Individuals are variable length binary strings.
- Each gene is called a codon. An 8 bit string.

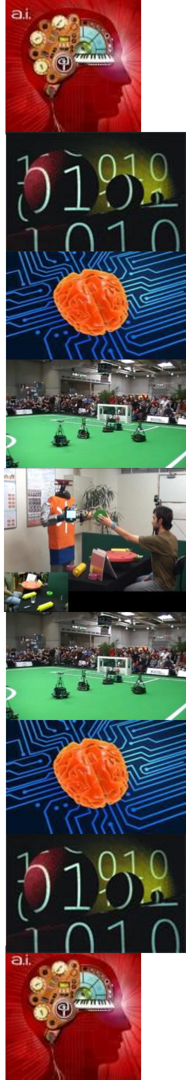




Grammatical Evolution Algorithm

Algorithm 1 Grammatical Evolution

Create an initial population of variable length binary strings
Map via a BNF grammar
  a) binary strings to expression using production rules
Evaluate fitness
do while {termination condition not met}
  Select fitter individuals for reproduction
  Recombine selected individuals
  Mutate offspring
  Evaluate fitness of offspring
  Replace all individuals in the population with offspring
end while
return best individual

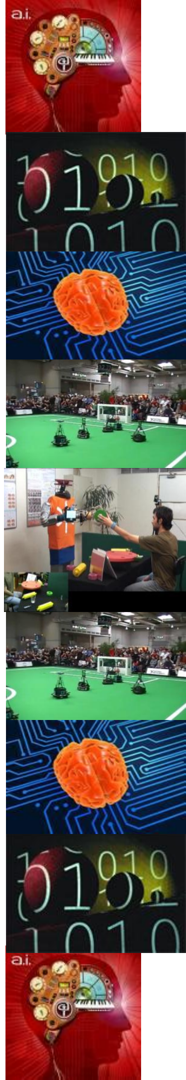


Initial Population Generation

- Randomly generate a population of variable length binary strings (individuals).

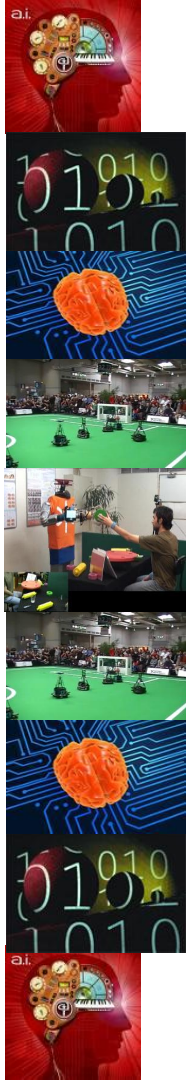
- Lengths are determined randomly from a lower bound and upper bound range

- The population size and the variable length limits are user specified. eg [8 - 20].



Mapping

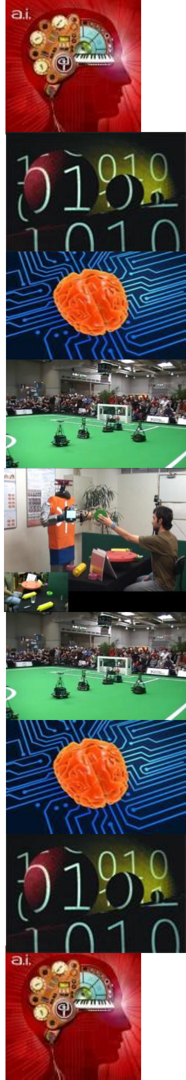
- A grammar with production rules needs to be specified.
- It must contain domain knowledge.
- Mapping involves converting the binary strings to decimal and using them to select the production rules to apply.
- Derivation i.e genotype to phenotype



Mapping Equation

Rule = (codon decimal value)%(No of production rules)

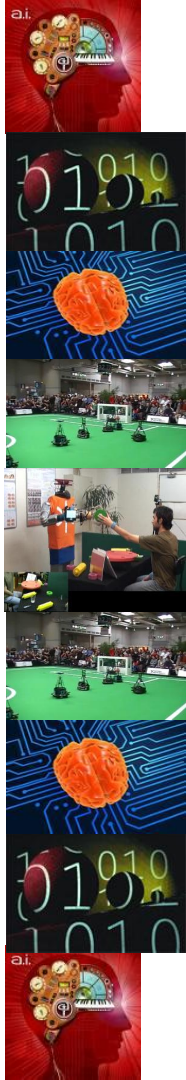
- A derivation tree (phenotype) is evolved by iterating and mapping through the sequence of codons.
- The derivation process is performed from left to right starting with the left-most non-terminal.



Mapping

- If the iteration process reaches the end of the sequence of codons before the derivation tree is evolved the procedure continues by looping to the start of the codon sequence, a process called *wrapping*.

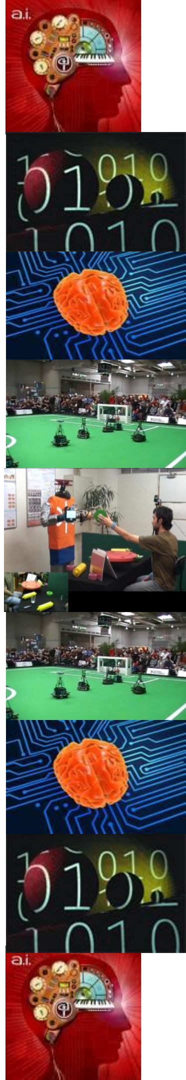
- The fitness of the phenotype is evaluated by applying it to a problem



Selection

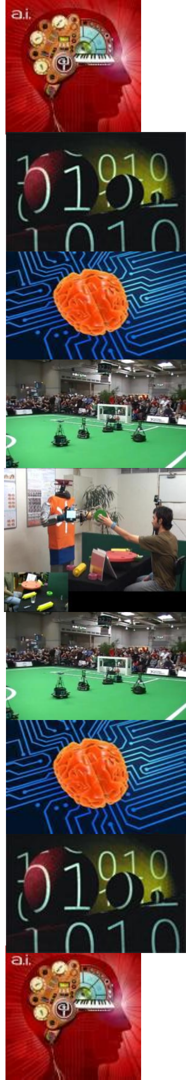
- Selection is applied in a similar manner as in genetic algorithms.

- Tournament selection or Fitness proportionate.



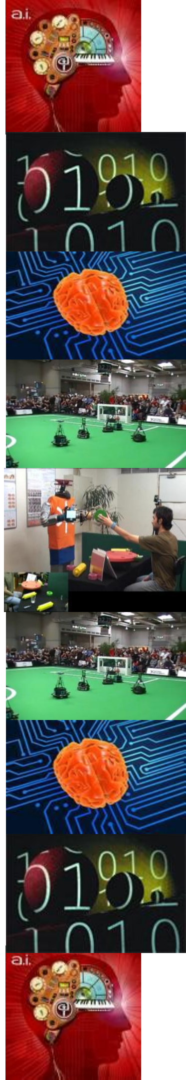
Genetic Operators

- Crossover
- Mutation
- Reproduction
- Elitism

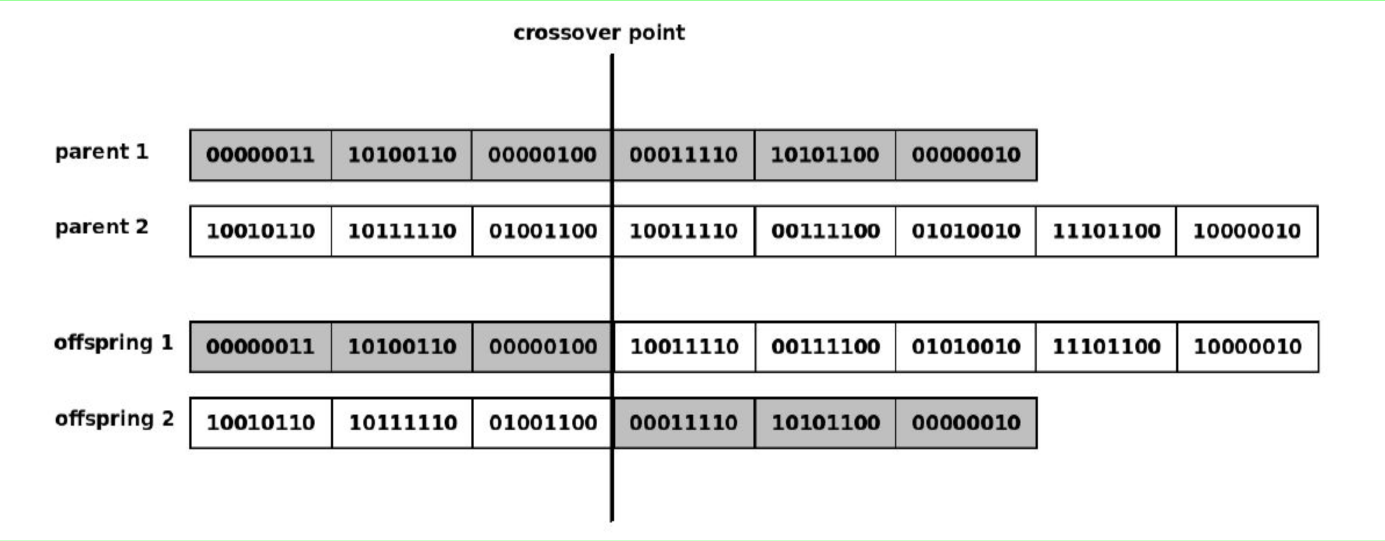


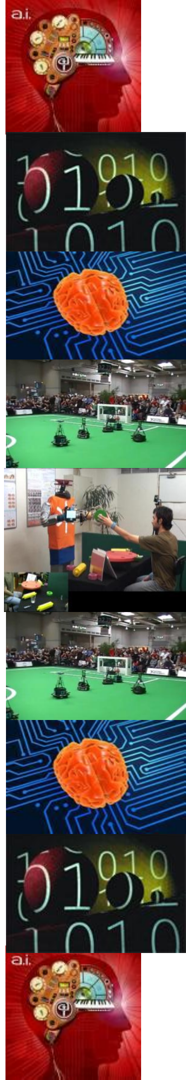
Crossover

- Single point crossover is the commonly used crossover method in GE.
- A crossover point is randomly selected from the shortest variable length parent.
- Other crossover methods are possible you need to consider the lengths



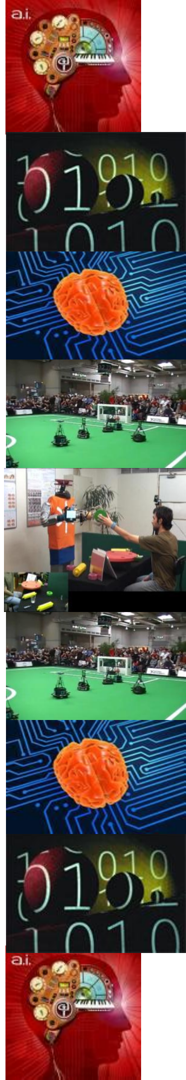
Crossover


crossover point
parent 1
00000011 10100110 00000100 00011110 10101100 00000010
parent 2
10010110 10111110 01001100 10011110 00111100 01010010 11101100 10000010
offspring 1
00000011 10100110 00000100 10011110 00111100 01010010 11101100 10000010
offspring 2
10010110 10111110 01001100 00011110 10101100 00000010


a.i.

Mutation, Pop replacement, Termination

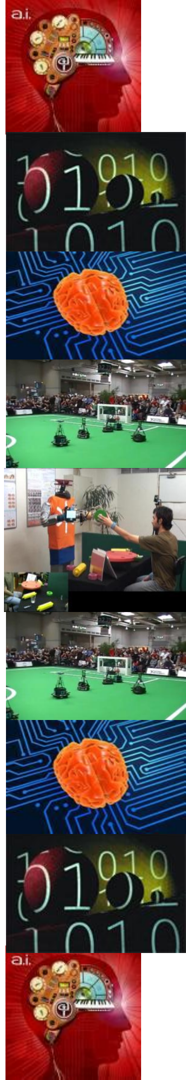
- Mutation - similar to GA i.e bit mutation.
- Population replacement is generational or steady state.
- Termination objective function is met or number of generations achieved.



Example

Each individual of the population is mapped into a phenotype using the **grammar** and the **production rule equation.**

Then its fitness is evaluated



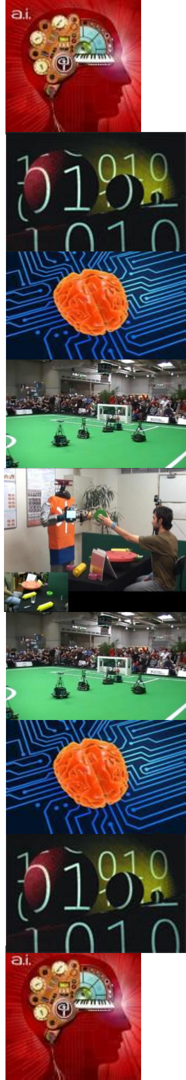
Example

Given the following individual (genotype) for a GE system that evolves pin numbers.

00000011	10100110	00000100	00011110	10101100	00000010

and the grammar.

<S> ::= <X><W><W> | <W><X><Y> | <X><W><Y>
<W> ::= <Y><X> | <Y><Z>
<X> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<Y> ::= a | b | c | d | e | f | g | h | i | j
<Z> ::= @| ~ | # | &|%| * |) | - | (| +

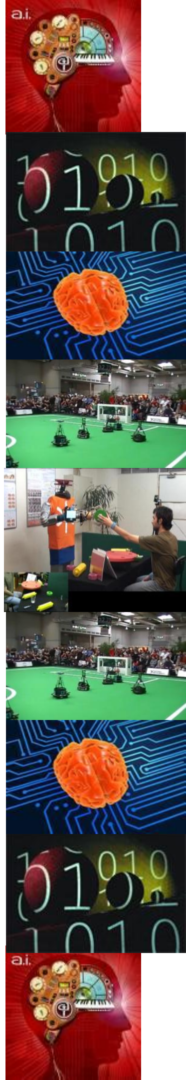


Context Free Grammar

(From earlier)

- A grammar **G = < N, T, P, S >**

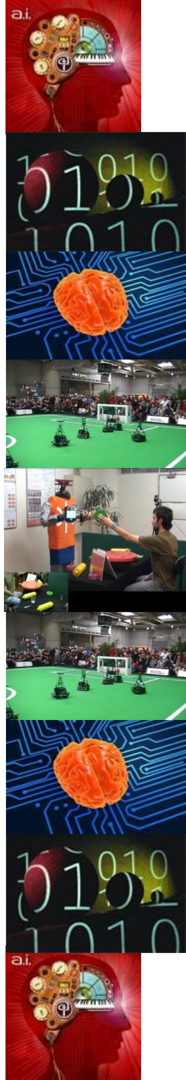
- N = non-terminals denoted by **< >** which means whatever is inside can be simplified or replaced by some other attributes.
- T - terminal final values.
- P- Production rules
- S - starting value



Grammar Explained

<S> ::= <X><W><W> | <W><X><Y> | <X><W><Y>
<W> ::= <Y><X> | <Y><Z>
<X> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<Y> ::= a | b | c | d | e | f | g | h | i | j
<Z> ::= @| ~ | # | &|%| * |) | - | (| +

A grammar G = < N, T, P, S >
N = <S>, <W>, <X> etc etc | means OR
T = 1, 2, a, b, @, *, % etc etc
P = production rules = <X><W>, 1, 2, a etc numbered from 0 to n.
S = starting S can be anything as defined by the user.

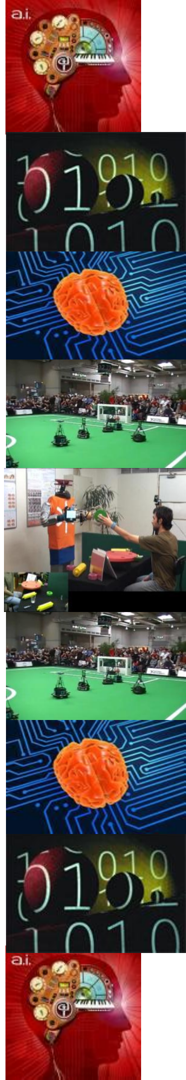


Mapping Example

We start by converting the binary codon to their respective decimal values.

00000011	10100110	00000100	00011110	10101100	00000010

5,166,4,30,172,2



Mapping

5,166,4,30,172,2

S = <S> // our starting point we need to simplify <S>

From the grammar we have 3 options viz

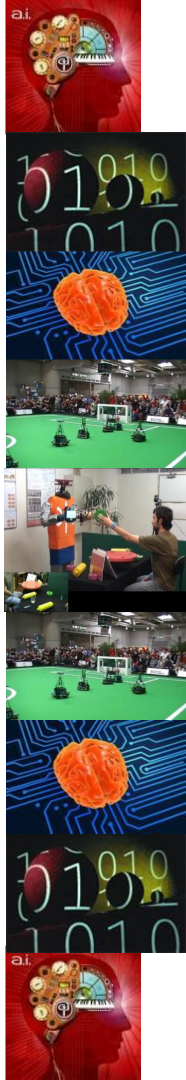
<S> ::= <X><W><W> | <W><X><Y> | <X><W><Y>

<W> ::= <Y><X> | <Y><Z>

<X> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

<Y> ::= a | b | c | d | e | f | g | h | i | j

<Z> ::= @ | ~ | # | & | % | * |) | - | (| +



Mapping

5,166,4,30,172,2

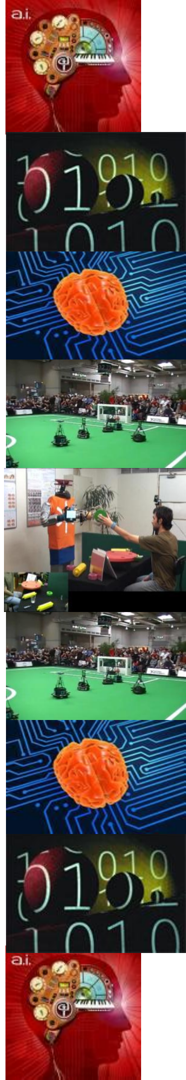
S = <S> // our starting point we need to simplify <S>

From the grammar we have 3 options viz

Production rule 0 - defined as <X><W><W> **OR**

Production rule 1 - defined as <W><X><Y> **OR**

Production rule 2 - defined as <X><W><Y>



Mapping

The choice of the rule to use it determined by the values of the GE individual and the **production rule equation** as follows:

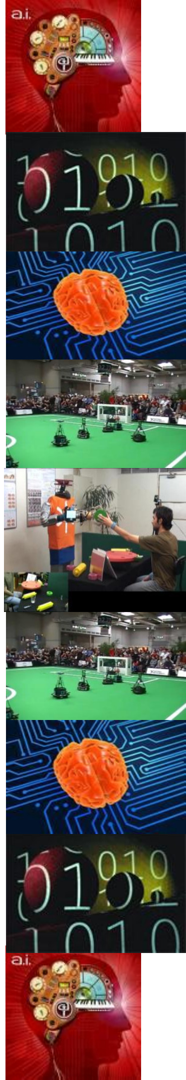
5,166,4,30,172,2

Rule = codon value % number of rules —prod equation

rule = 5 % 3 = 2 (5 codon value, 3 rules, select rule 2)

pin = <S> // r 2 = <X><W><Y>

pin = <X><W><Y>



Mapping

pin = <X><W><Y>

5,166,4,30,172,2

We consider & simplify the left most non-terminal.

<X> has 10 terminal rules (rule 0 - rule 9)

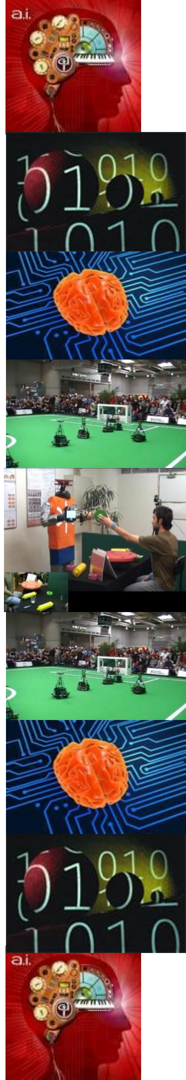
<S> ::= <X><W><W> | <W><X><Y> | <X><W><Y>

<W> ::= <Y><X> | <Y><Z>

<X> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

<Y> ::= a | b | c | d | e | f | g | h | i | j

<Z> ::= @| ~ | # | &|%| * |) | - | (| +



Mapping

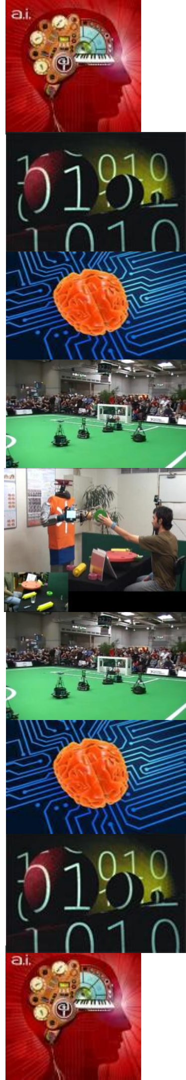
pin = <X><W><Y>

5,166,4,30,172,2

We consider & simplify the left most non-terminal.

rule for <X> = 166 % 10 = 6 i.e prod rule $6 \longrightarrow 7$

pin = 7<W><Y> // 7 is a terminal so we move to the next non-terminal <W>



Mapping

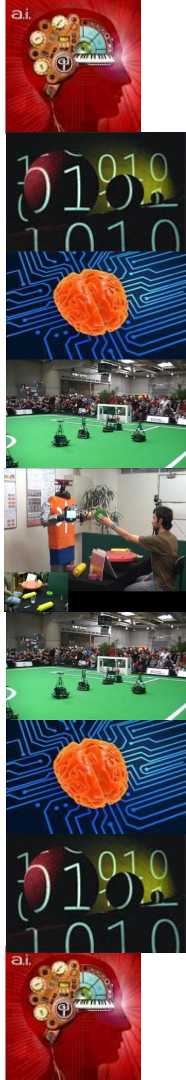
pin = 7<W><Y>

5,166,4,30,172,2

We consider & simplify the left most

non-terminal which is <W>. It has 2 rules in the grammar which are <Y><X> and <Y><Z>

<S> ::= <X><W><W> | <W><X><Y> | <X><W><Y>
<W> ::= <Y><X> | <Y><Z>
<X> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<Y> ::= a | b | c | d | e | f | g | h | i | j
<Z> ::= @| ~ | # | &| %| * |) | - | (| +



Mapping

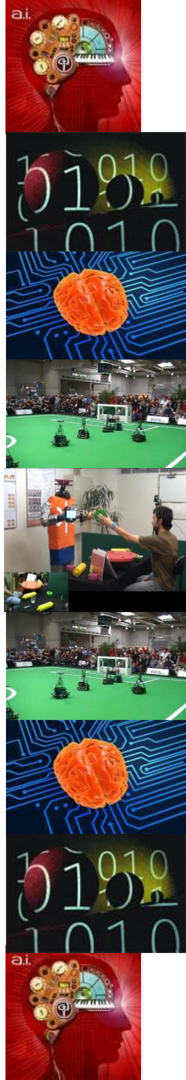
pin = 7<W><Y>

5,166,4,30,172,2

We consider & simplify the left most non-terminal

rule for <W> = 4 % 2 = 0 i.e prod rule 0 → <Y><X>

pin = 7<Y><X><Y> // we consider the left most non-terminal <Y>



Mapping

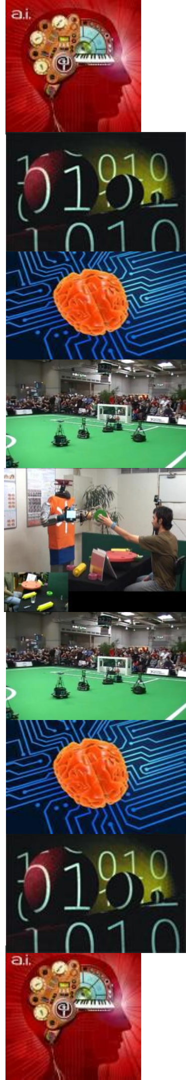
pin = 7<Y><X><Y>

5,166,4,30,172,2

We consider & simplify the left most non-terminal

rule for <Y> = 30 % 10 = 0 i.e prod rule 0 → a

pin = 7 a <X><Y> // a is a terminal, next is <X>



Mapping

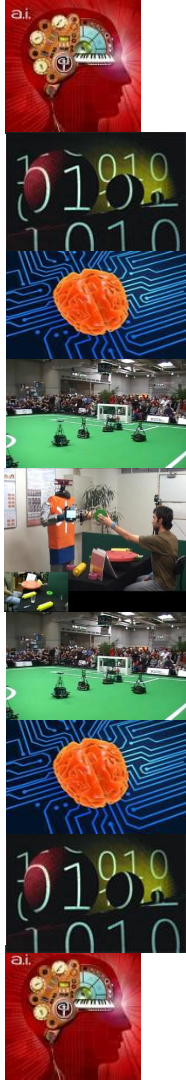
5,166,4,30,172,2

pin =7 a <X><Y>

We consider & simplify the left most non-terminal

rule for <X> = 172 % 10 = 2 i.e prod rule $2 \longrightarrow 3$

pin = 7 a 3 <Y> // 3 is a terminal, next is <Y>



Mapping

pin =7 a 3 <Y>

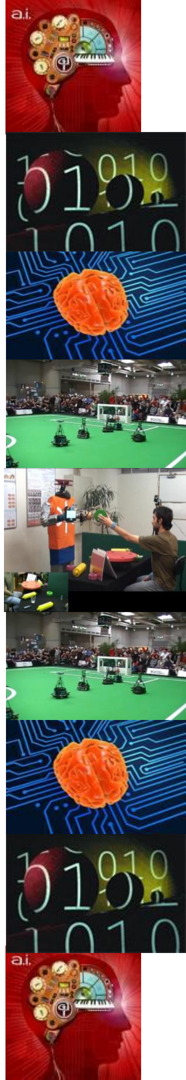
5,166,4,30,172,2

We consider & simplify the left most non-terminal

rule for <Y> = 2 % 10 = 0 i.e prod rule 0 → a

pin = 7 a 3 a // this is the pin number associated with that GE genotype.

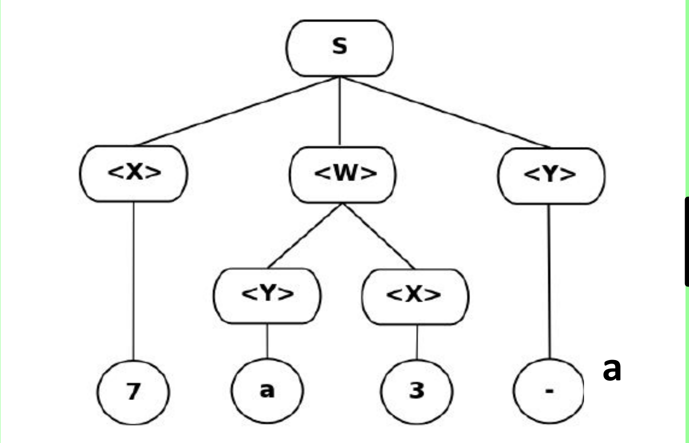
The fitness of the pin number can then be evaluated.



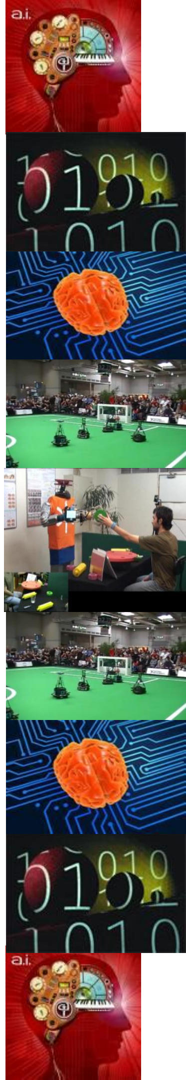
Mapping

5	166	4	30	172	13

7 a 3 a



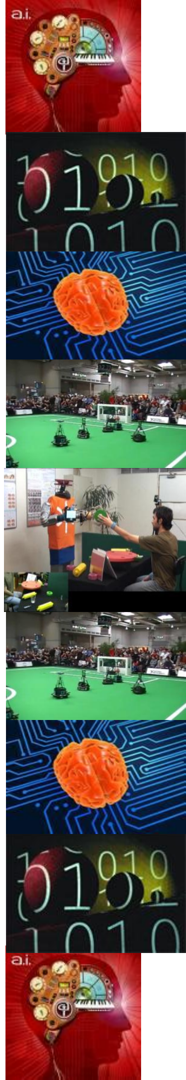
Derivation tree



Mapping

Algorithm continues with the variation of individuals

If the codons run out before derivation is complete there is looping back which is called wrapping.



Artificial Ant Food Search

$$N = \{code, line, expr, if - statement, op\}$$

$$T = \{left(), right(), move(), food\_ahead(), else, if, \{, \}, (,), ;\}$$

$$S = code$$

And P can be represented as:

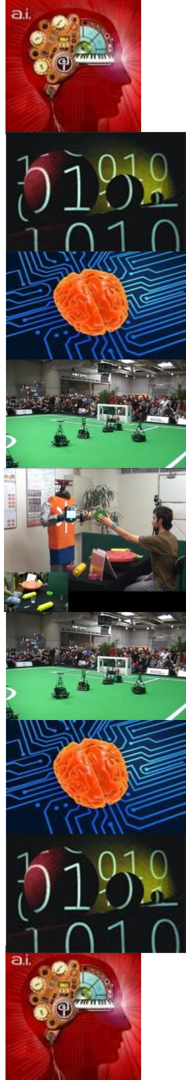
```
(1) <code> : : =  <line>                (0)
                 |<code><line>          (1)

(2) <line> : : =  <if-statement>        (0)
                 |<op>                  (1)

(3) <if-statement> : : = if(food_ahead())
                            {<line>}
                         else
                            {<line>}

(4) <op> : : =    left();                (0)
                | right();               (1)
                | move();                (2)
```

```
move();
left();
if(food_ahead())
    left();
else
    right();
right();
if(food_ahead())
    move();
else
    left();
```

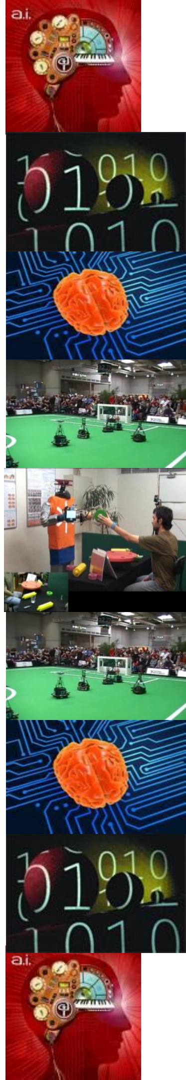


Applications of Genetic Programming (GP)

<u>Symbolic Regression</u>

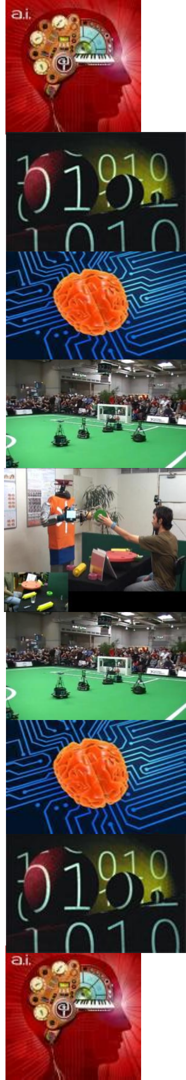
In this example, GP will attempt to evolve a program that, given an input value, produces an output value. The creation of a function that matches a collection of input/output values is known as **symbolic regression**.

The collection is known as **fitness cases**



Fitness Cases

Case No	x	y	Case No	x	y
1	2	6	11	-2	2
2	4	20	12	-4	12
3	6	42	13	-6	30
4	8	72	14	-8	56
5	10	110	15	-10	90
6	11	132	16	-11	110
7	12	156	17	-12	132
8	13	182	18	-13	156
9	14	210	19	-14	182
10	15	240	20	-15	210



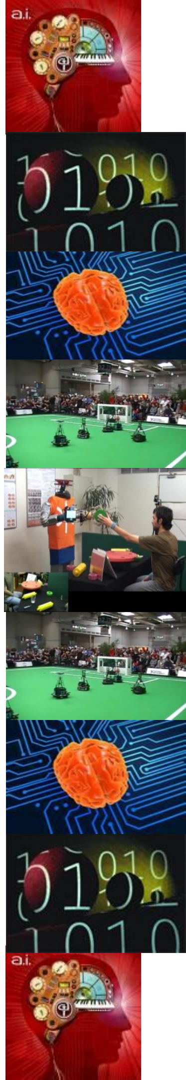
SR (GP)

This data set shows the value of **y** for various **x** values.

This is an example of regression, rather than classification.

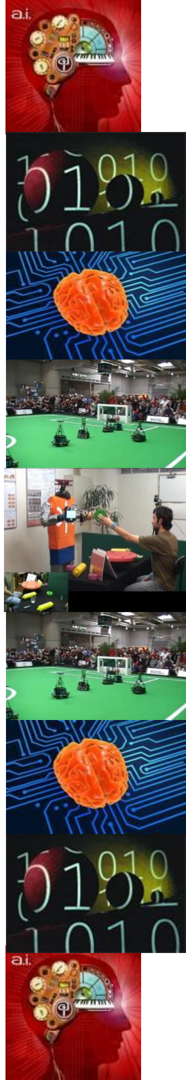
Regression problems seek to predict a numeric outcome for

a given input.



SR (GP)

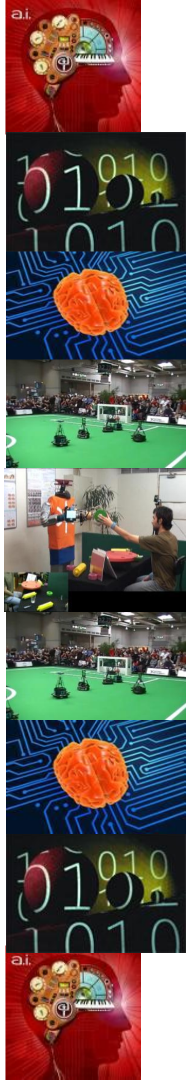
The fitness function, being the only real access the system has to the problem, must be carefully designed, with a maximum amount of discrimination possible, for a given runtime.



SR (GP) - Fitness Function

In symbolic regression, the only explicit knowledge of the target function that the fitness metric has access to is a table of x values and corresponding y values.

- Can we use a function that tell us how many of the fitness cases the program got right ?
- What would be the ideal fitness function ?

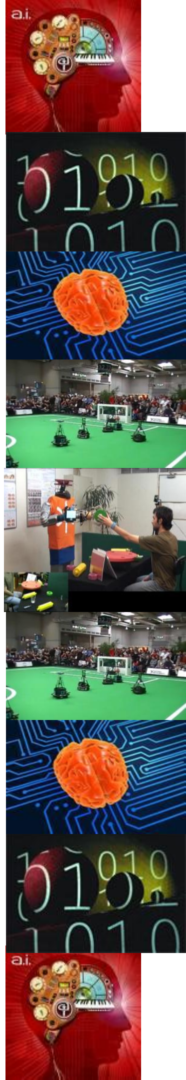


SR (GP)- Fitness Function

- Since both the fitness function and the fitness cases are numeric.

- We can use a function that sums up the difference between the actual value and the expected values for all the fitness cases.

- The overall fitness will represent the error margin for all the fitness cases.



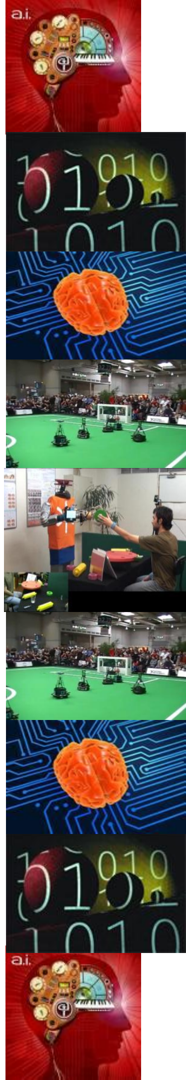
SR (GP) - Terminal & Function Sets

<u>Terminal</u>

Input is known to be variable x thus **{ x }**

<u>Function</u>

{+,*,-, /(protected)**}**

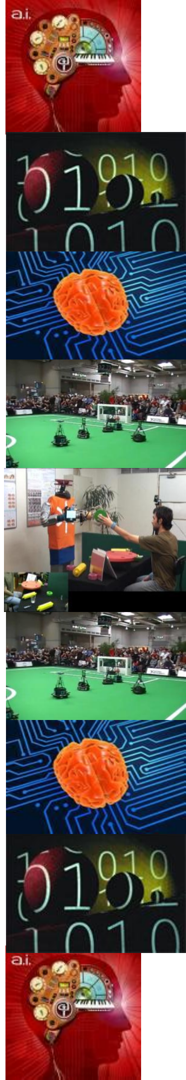


SR (GP)- Operators

Crossover.

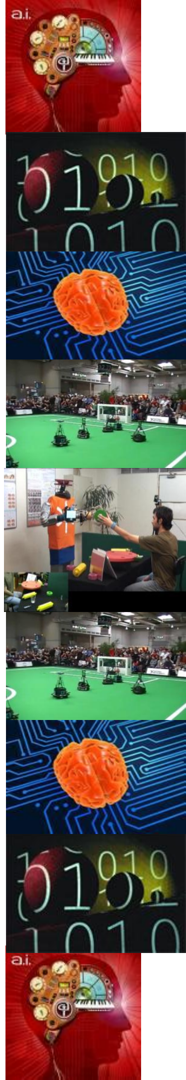
Mutation.

Reproduction.

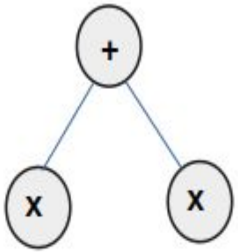


SR (GP)- Template of Parameters

Parameter	Value
Population size	1000
Initial tree generation	ramped half-and-half
Initial tree depth	6
Max offspring depth	4
Selection method	tournament
Tournament size	20
Function set	+ , - , * ,/
Crossover rate	80%
Mutation rate	20%
Mutation type	point
Mutation offspring depth	4
Fitness function	accuracy
Maximum generations	100

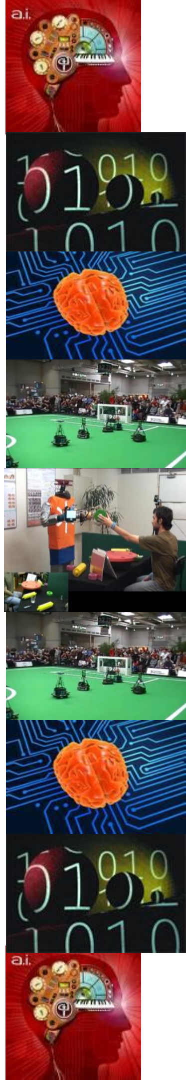


SR (GP)- Output

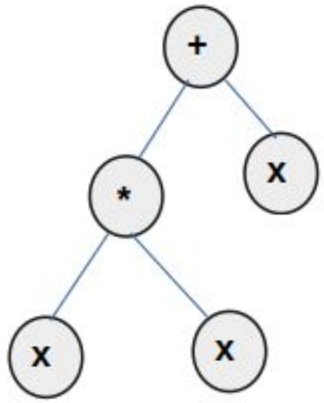


f(y) = x + x

Fitness case 1 : f(y) = 2 + 2 = 4 Expected 6 -> acc_error = 2
Fitness case 2 : f(y) = 4 + 4 = 8 Expected 20 -> acc_error = 14
Fitness case 3 : f(y) = 12 Expected 42 -> acc_error = 44
.
.
.
Fitness = acc_error.

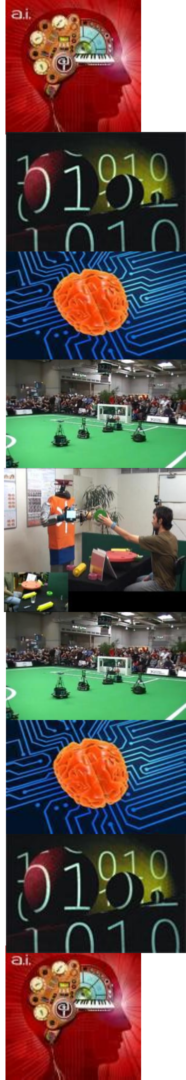


SR (GP)- Output



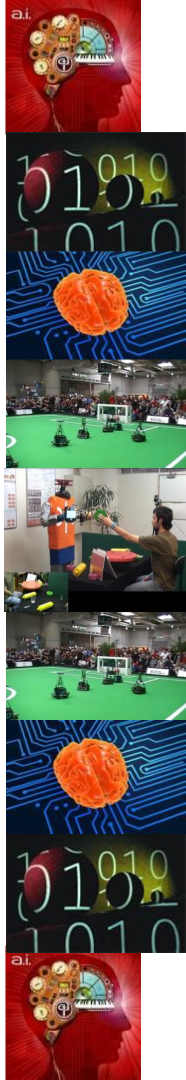
f(y) = x*x + x

Fitness case 1 : f(y) = 4 + 2 = 6 Expected 6 -> acc_error = 0
Fitness case 2 : f(y) = 20 Expected 20 -> acc_error = 0
Fitness case 3 : f(y) = 42 Expected 42 -> acc_error = 0
.
.
.
Fitness = acc_error = 0



SR (GP)- Generations

```
Iteration: 1, Current error = 20710.295679925002, Best Solution
    Length = 20
Iteration: 2, Current error = 20710.295679925002, Best Solution
    Length = 20
Iteration: 3, Current error = 20710.295679925002, Best Solution
    Length = 20
Iteration: 4, Current error = 18435.519210663904, Best Solution
    Length = 16
Iteration: 5, Current error = 18435.519210663904, Best Solution
    Length = 16
...
Iteration: 996, Current error = 8.510634781265793, Best Solution
    Length = 14
Iteration: 997, Current error = 8.510634781265793, Best Solution
    Length = 14
Iteration: 998, Current error = 8.510634781265793, Best Solution
    Length = 14
Iteration: 999, Current error = 8.510634781265793, Best Solution
    Length = 14
```



QUESTIONS