Cartesian Genetic Programming

Author: Michaela Šikulová

Genetic Programming

Evolutionary computation
- Non-deterministic search algorithms
- Based on aspects of Darwin's theory of evolution
- 1970s

Alan Turing (1948)
- Idea of artificial evolution

Genetic programming (GP)
- Automatic evolution of computer programs
- Generating random programs (initial population)
- Evaluation of each program in population (determining fitness)
- Creating of new generation (using recombination, mutation)
- Till searched program is found

Variants of GP

Tree-Based GP
- LISP expressions
- Unique path between any pair of nodes

Gramatical evolution (GE)
- Grammar defined using Backus-Naur form (BNF)

Push GP
- Lee Spector – stack-based computer language Push

Cartesian Graph-Based GP
- Graphs allow more than one path between any pair of nodes

Cartesian Genetic Programming

Grew from a method of evolving digital circuits (Miller et al., 1997)

Programs represented as directed acyclic graphs

Applications:

- Self-modiffying digital circuits
- Evolution of Electronic Circuits
- Image processing
- Artificial art and creativity
- Medical applications

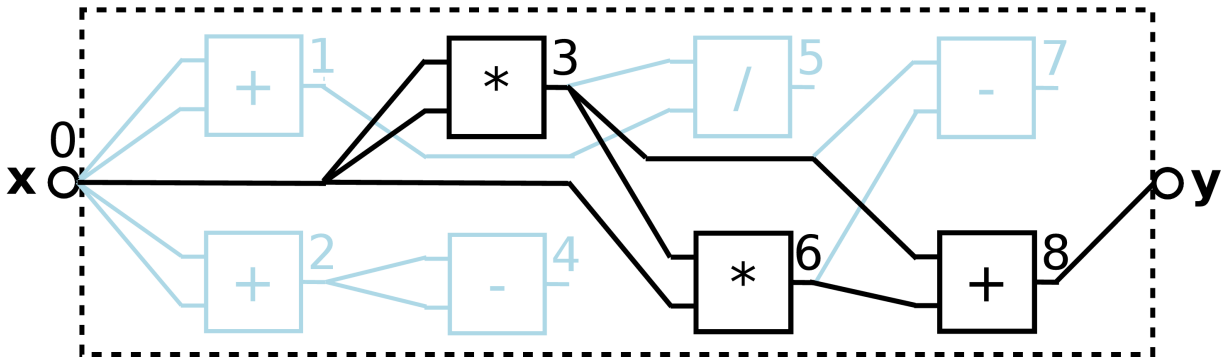
Cartesian Program (CP)

Directed acyclic graph

- More general than tree - two directed paths from a single starting node meet back at the same ending node

Two-dimensional grid of computation nodes

- Coding nodes, non-coding nodes
- Similar to digital circuits



Cartesian Program (CP)

A cartesian program is a 9-tuple

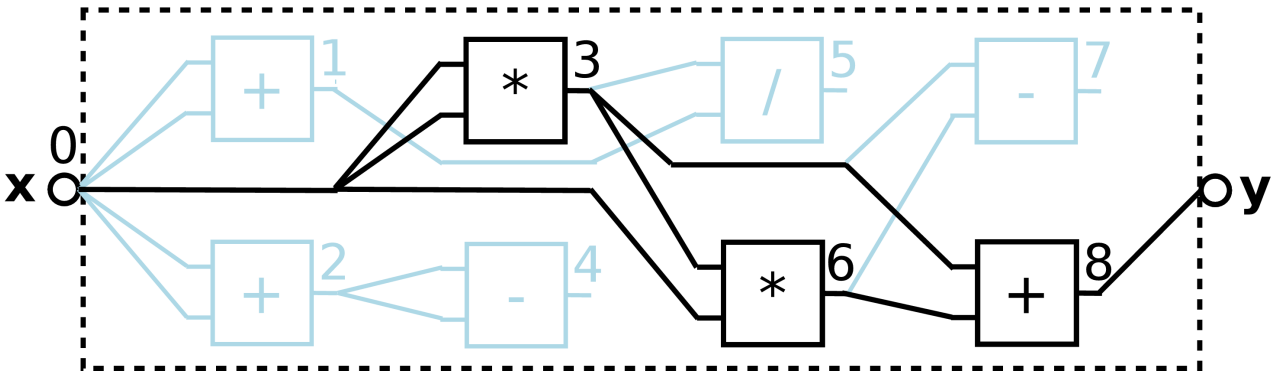
$$P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$$

where

G	represents genotype as a sequence of integers
n_i	number of program inputs
n_o	number of program outputs
n_n	number of node input connections
F	set of node functions
n_f	number of node functions
n_r	number of rows
n_c	number of columns
l	levels back parameter (how many columns of cells may have their outputs connected to a node in current column)

Cartesian Program (CP)

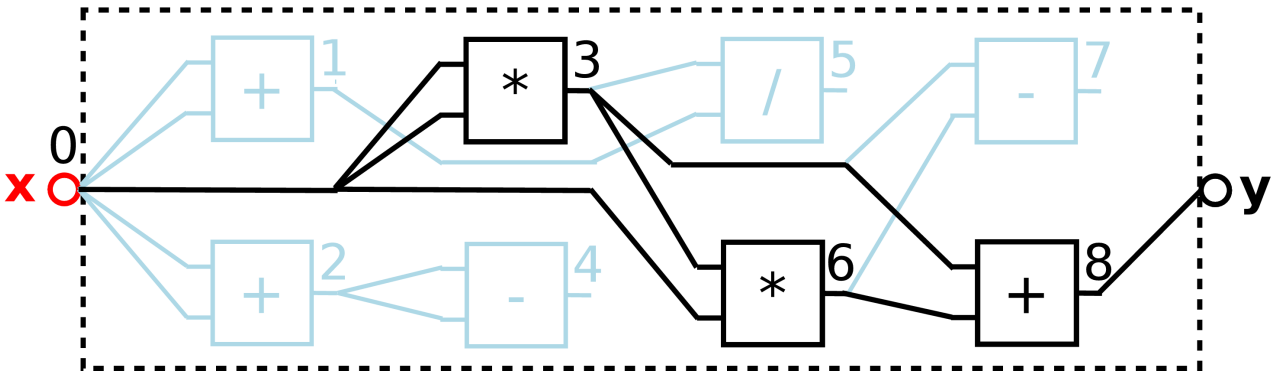
Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Program (CP)

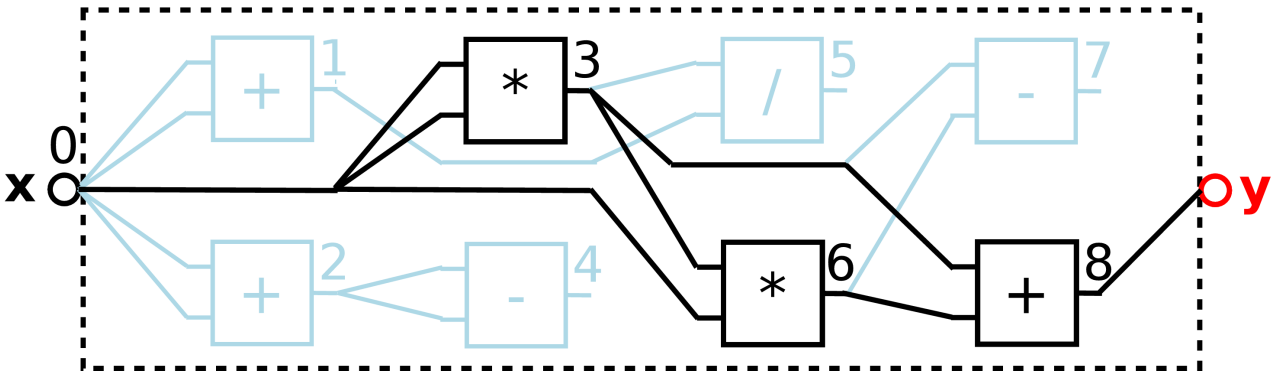
Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Program (CP)

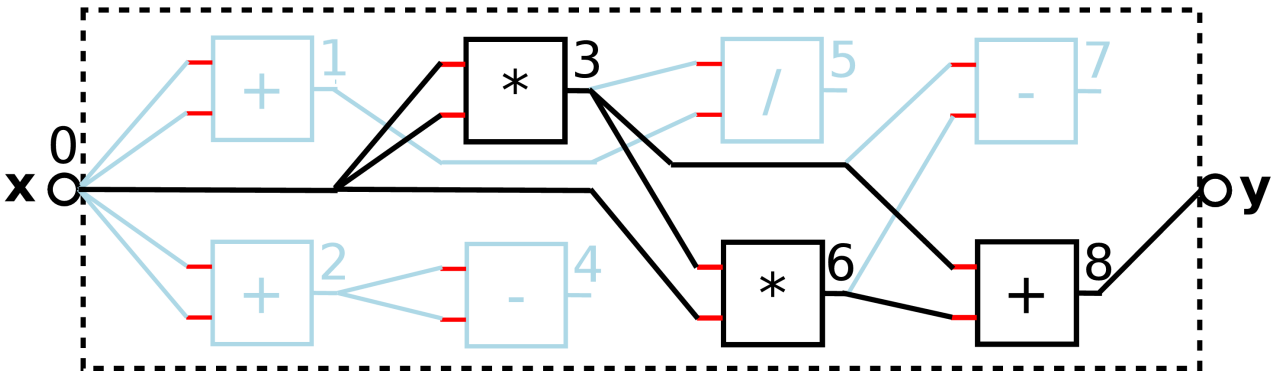
Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Program (CP)

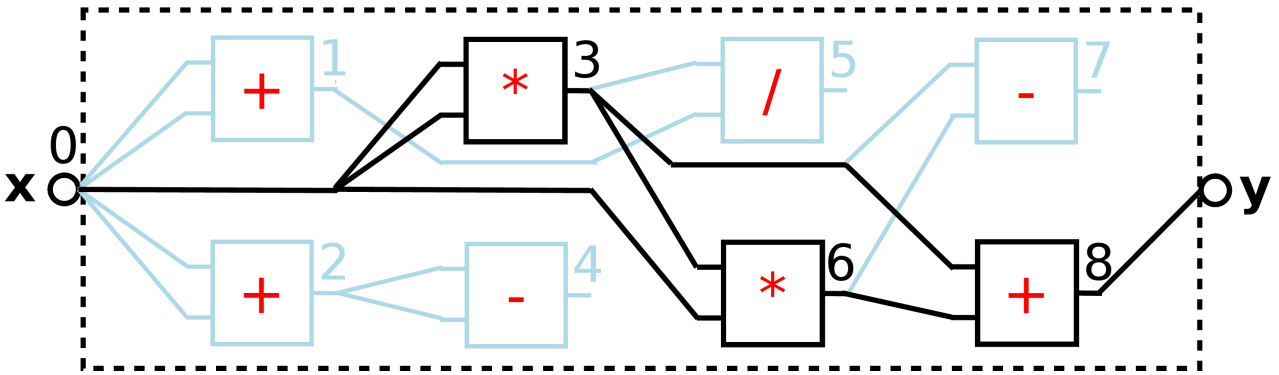
Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Program (CP)

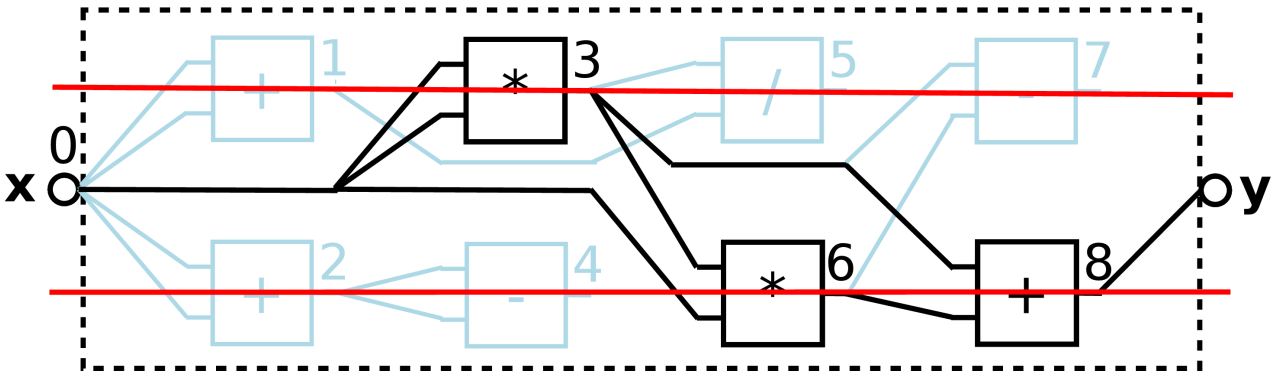
Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Program (CP)

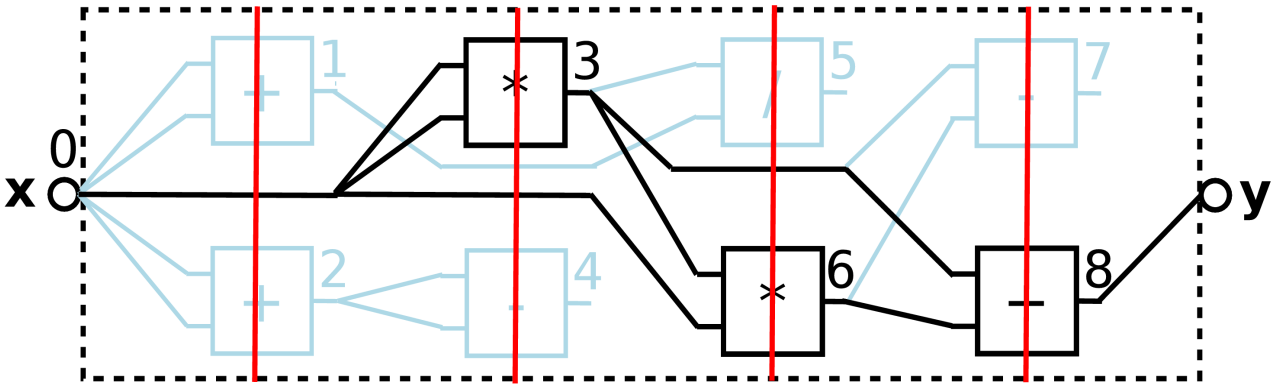
Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Program (CP)

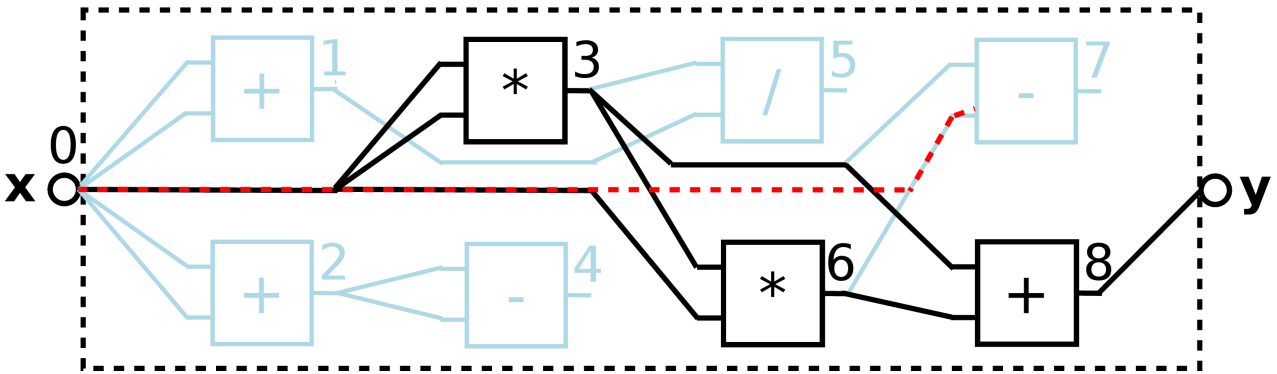
Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Program (CP)

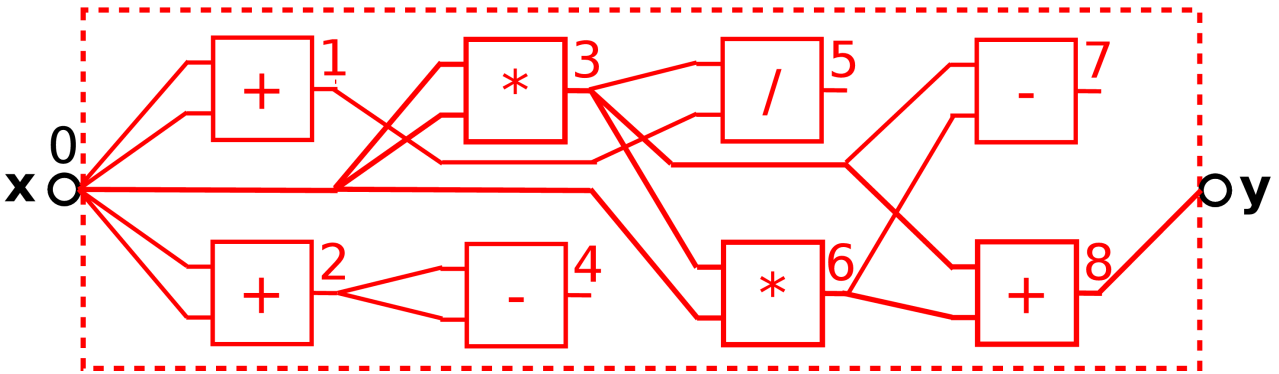
Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Program (CP)

Let $P = (G, n_i, n_o, n_n, F, n_f, n_r, n_c, l)$ be a cartesian program, where
G = (0, 0, 1, 0, 0, 1, 0, 0, 3, 2, 2, 2, 3, 1, 4, 3, 0, 3, 3, 6, 2, 3, 6, 1, 8),
$n_i = 1$, $n_o = 1$, $n_n = 2$, $F = \{+, -, *, /\}$, $n_f = 4$, $n_r = 2$, $n_c = 4$, $l = 4$:



$$y = x^2 + x^3$$

Cartesian Genetic Programming

Training data set

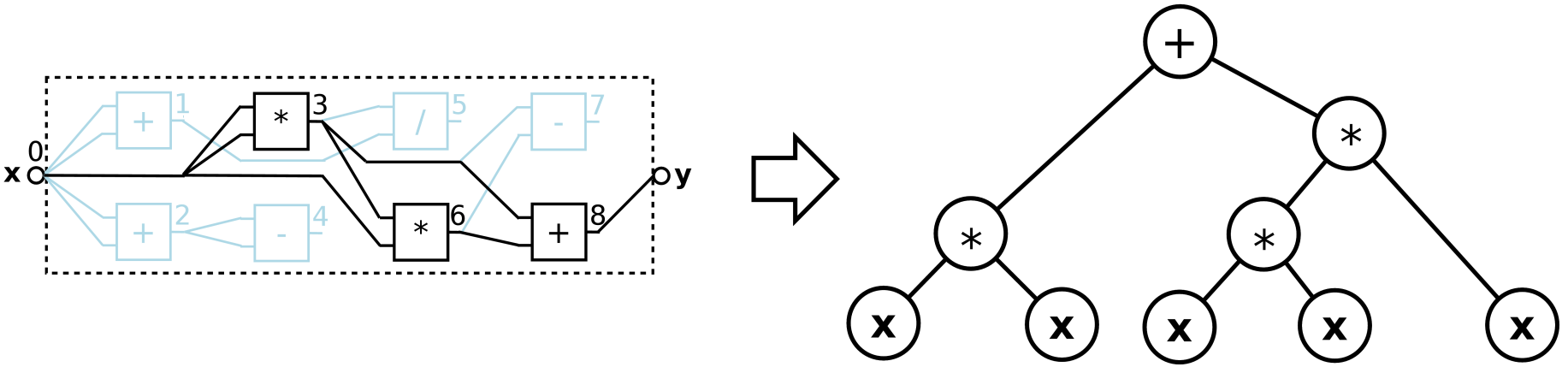
- CGP training data set (TDS) is a set of n-tuples, where n = $n_i + n_o$.
- Cardinality of TDS in CGP typically goes from tens to ten thousands in dependence on application domain.
- A one-bit full adder TDS has 8 5-tuples
- Symbolic regression f(x) with 200 training data points has 200 2-tuples

CP fitness evaluation

- A fitness evaluation is an assessment used to determine current fitness level of CP.
- CP fitness evaluation includes runing CP for every n-tuple in TDS (interpretation of CP).

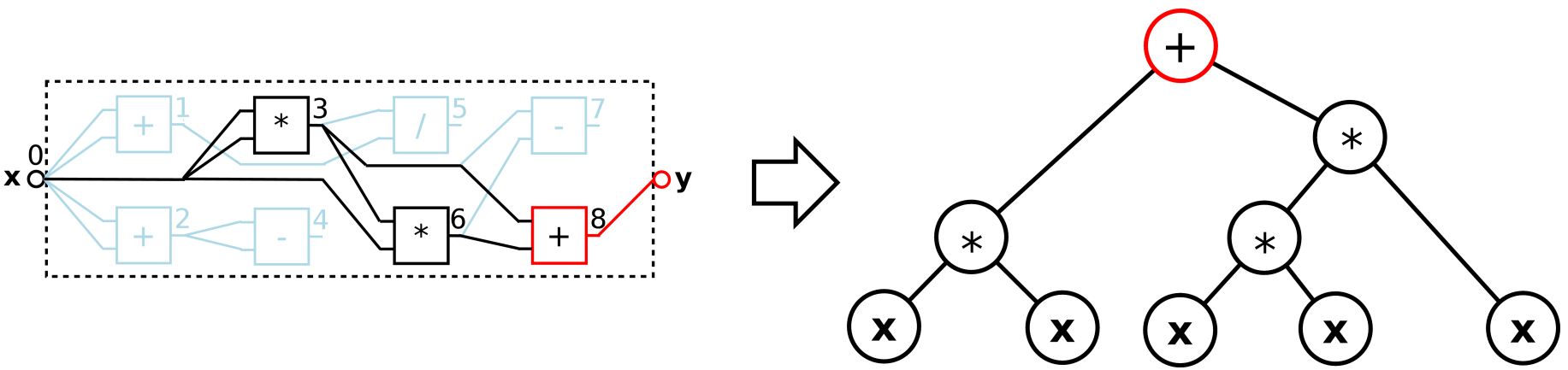
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



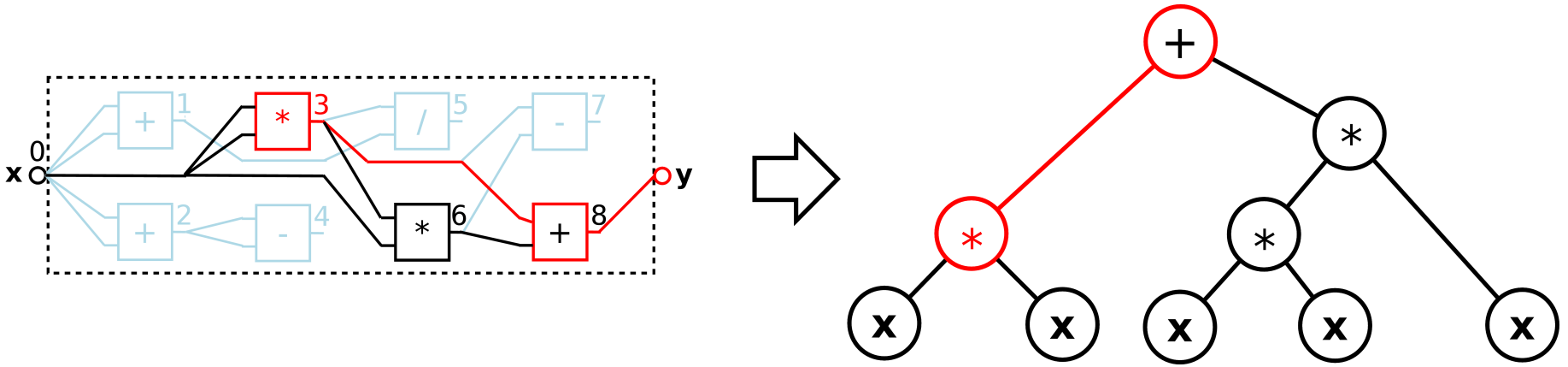
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



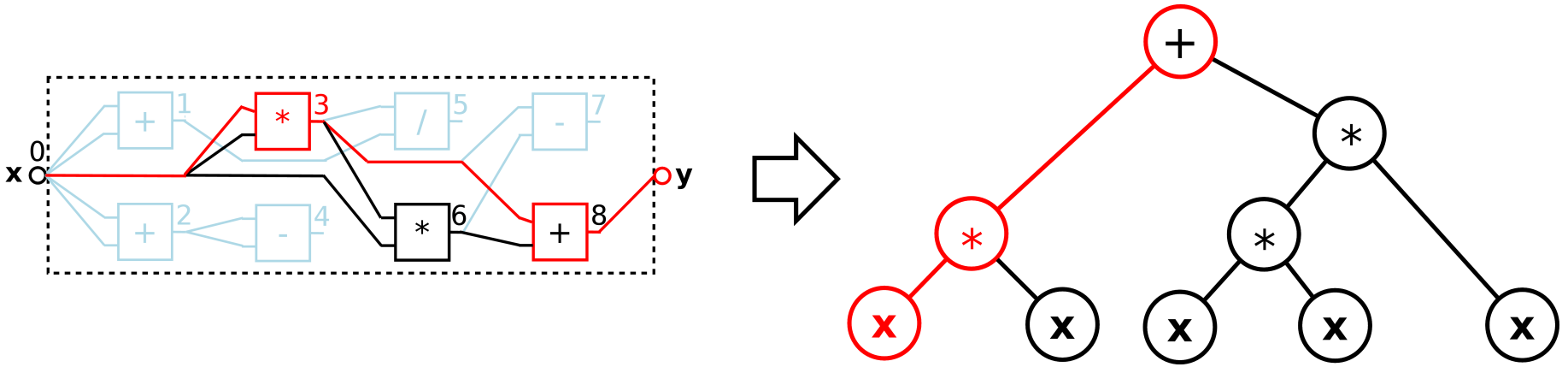
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



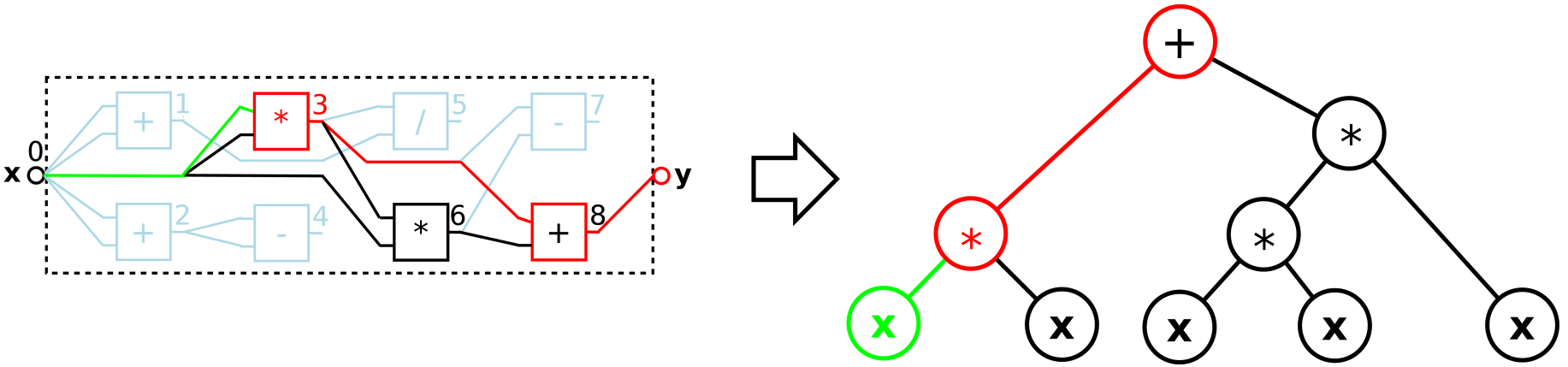
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



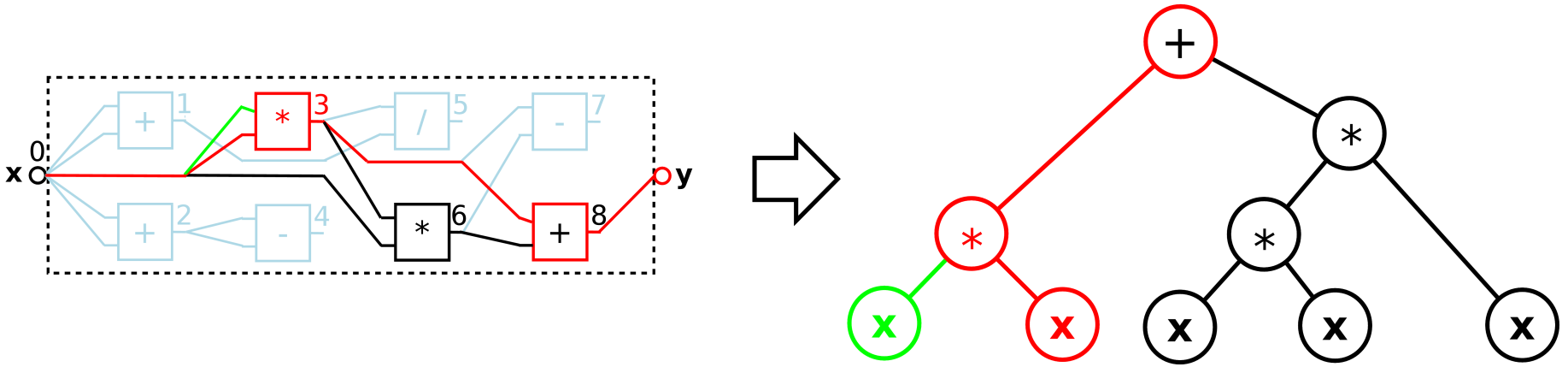
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



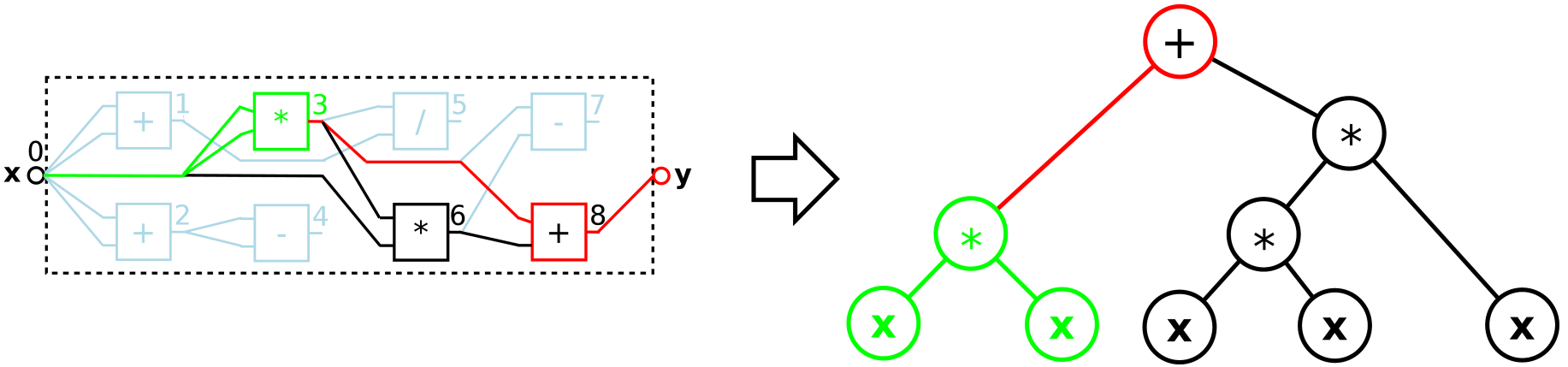
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



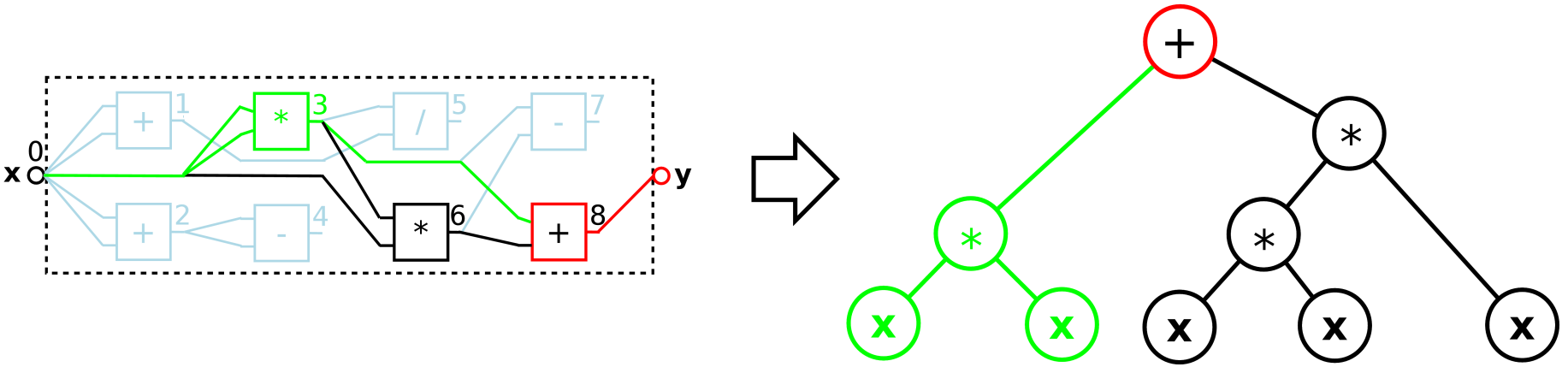
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



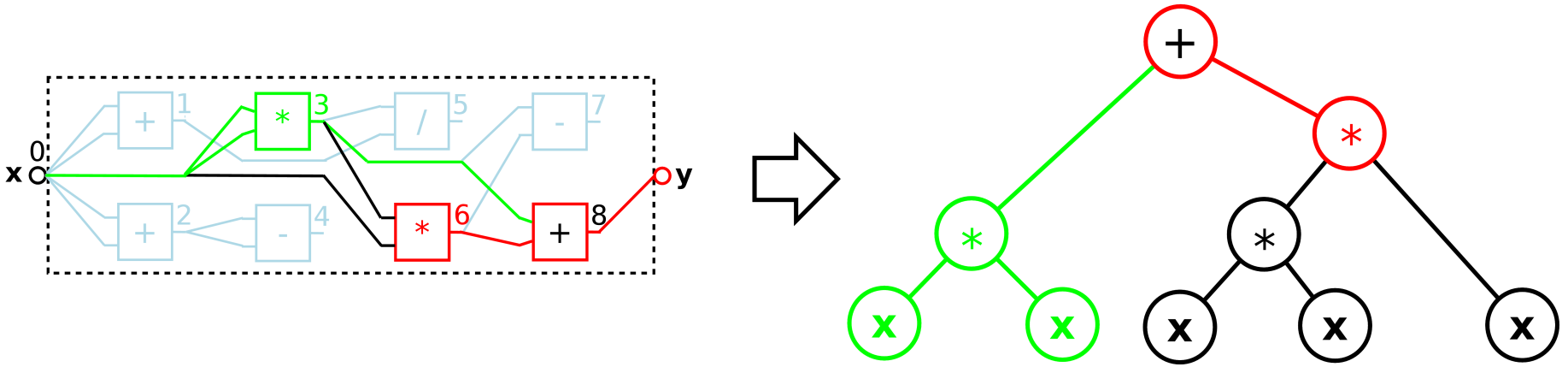
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



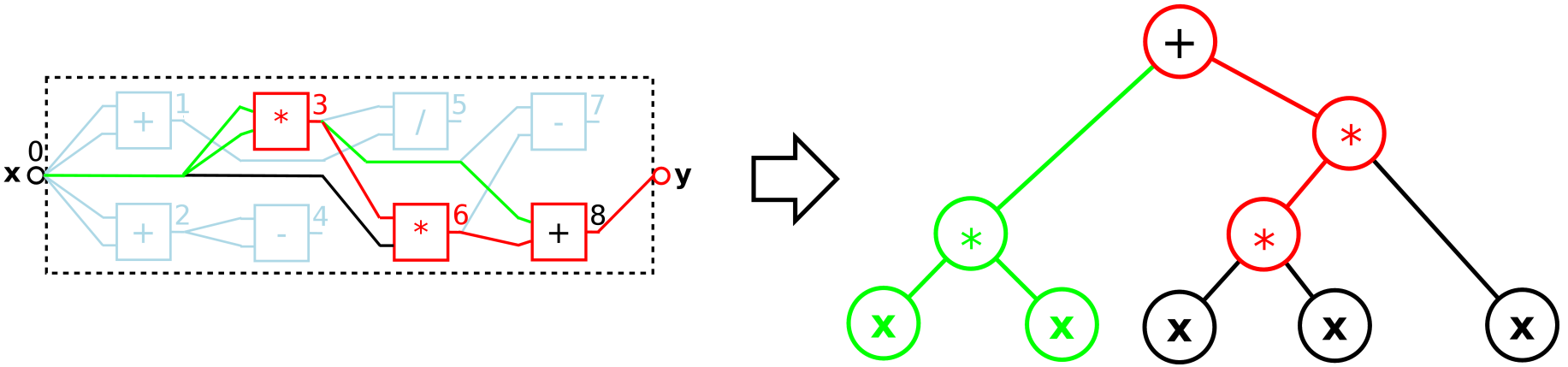
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



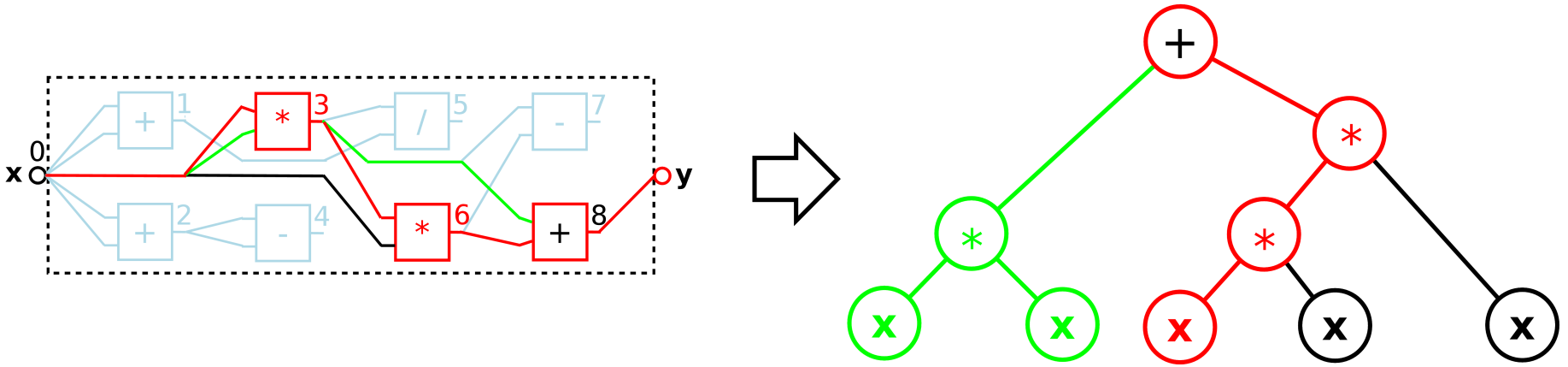
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



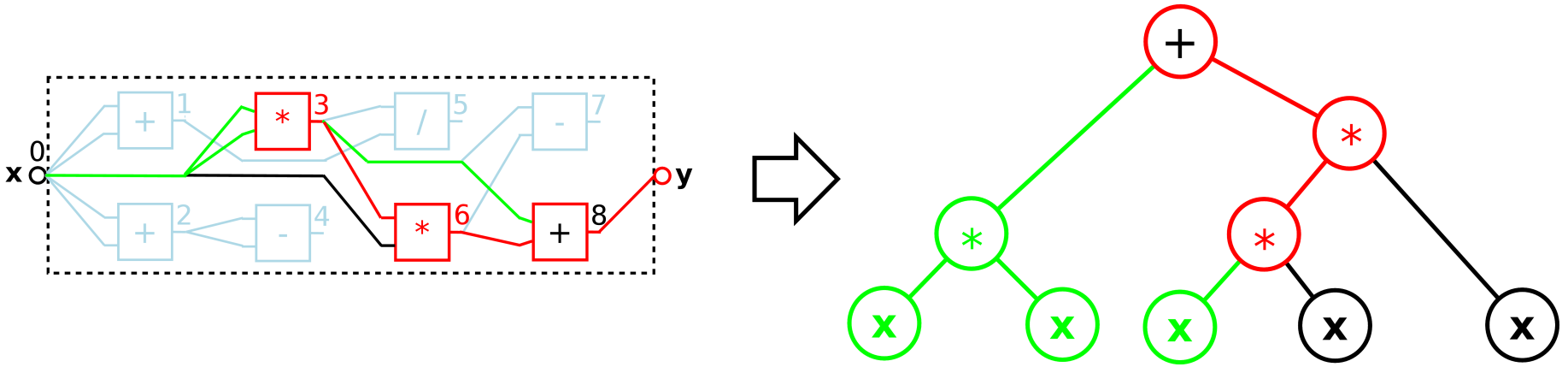
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



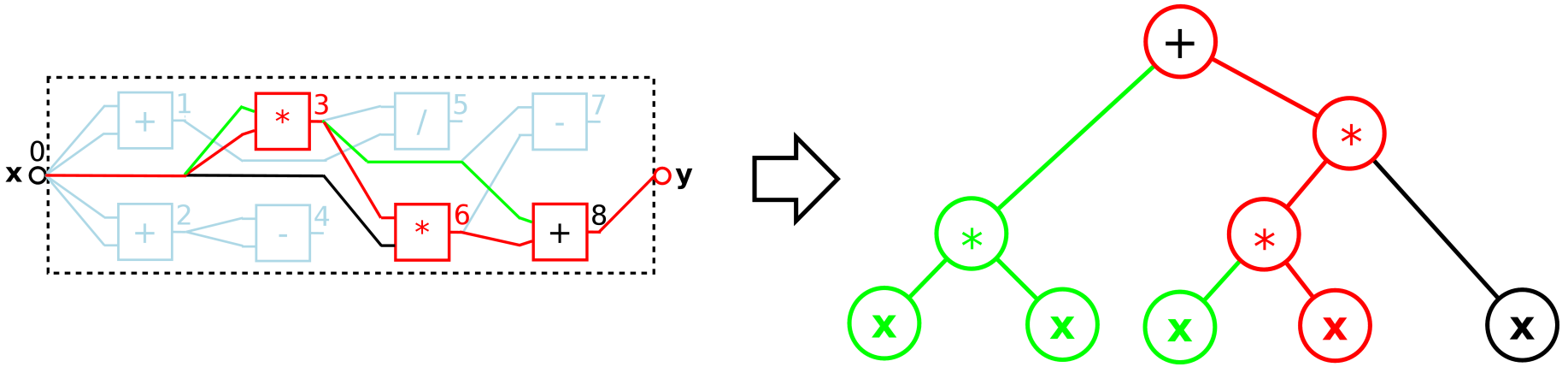
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



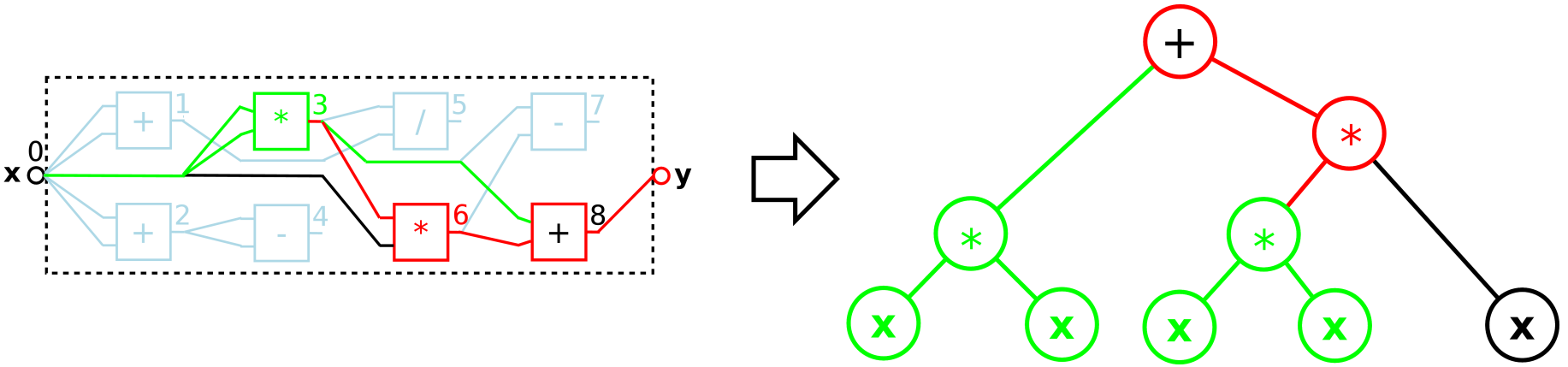
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



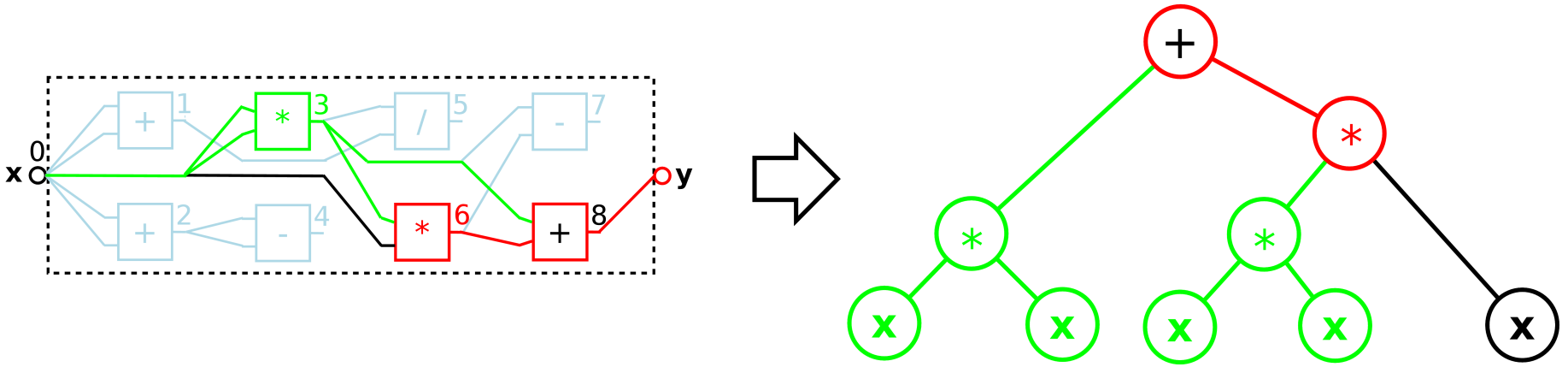
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



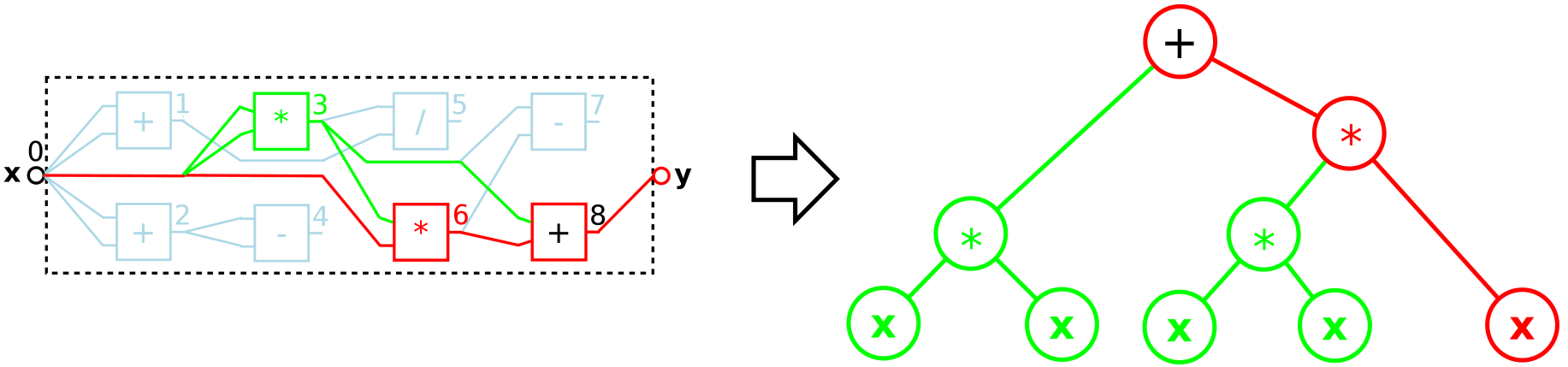
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



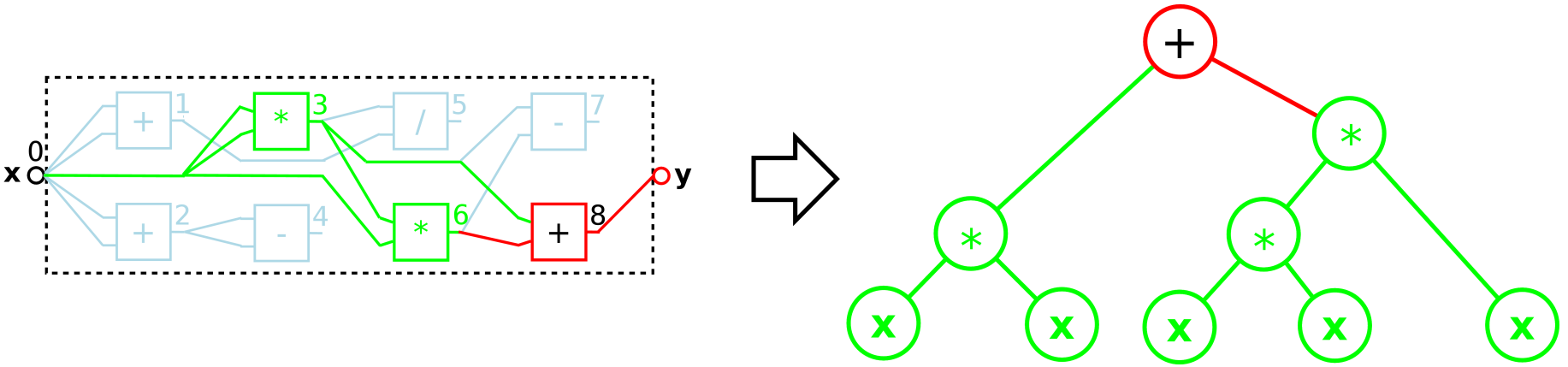
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



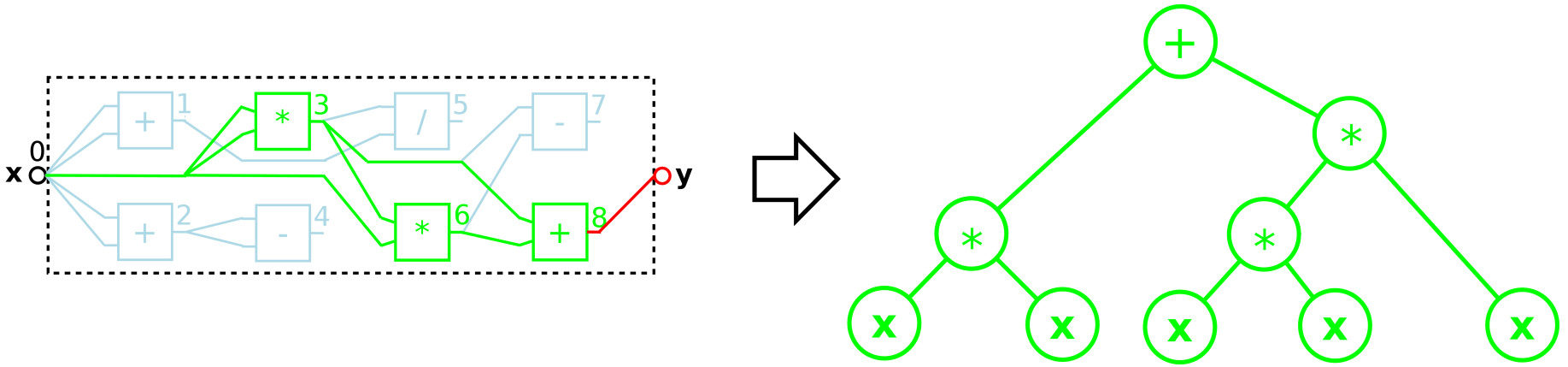
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



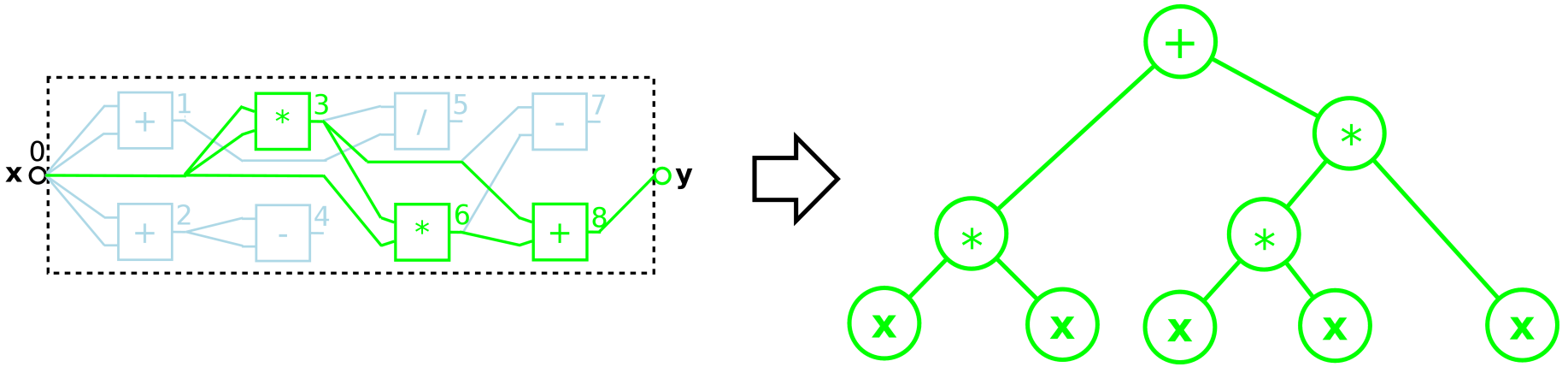
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



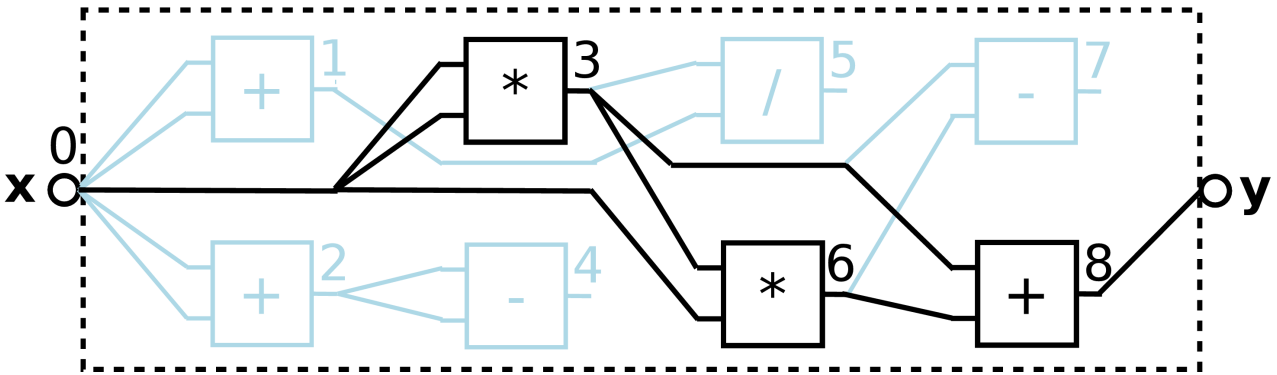
CP Interpretation – Recursive Descent Interpretation

- Similar to tree representation
- + Only use active nodes (phenotype)
- - nodes can be evaluated many times



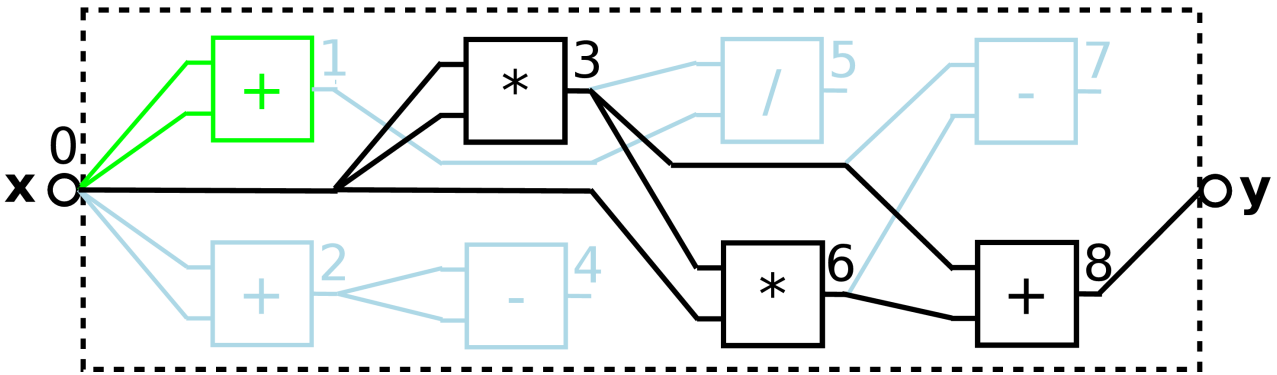
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



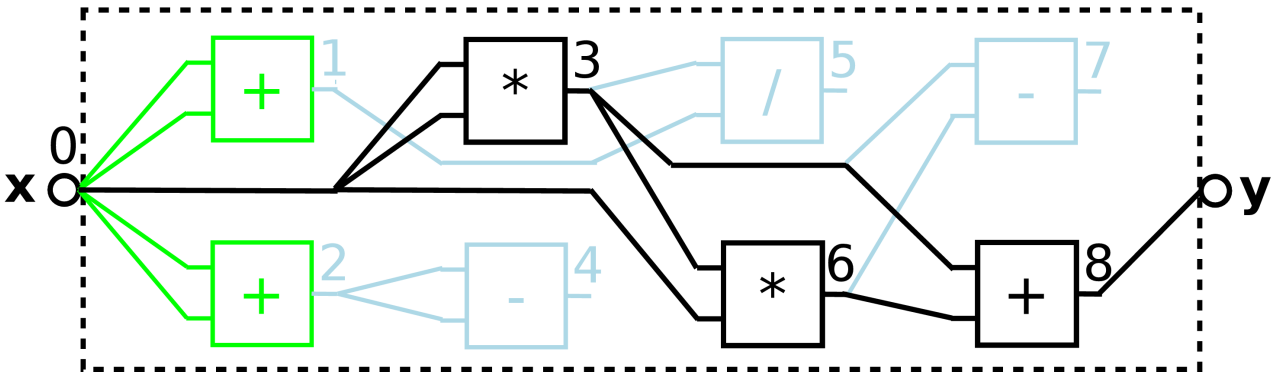
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



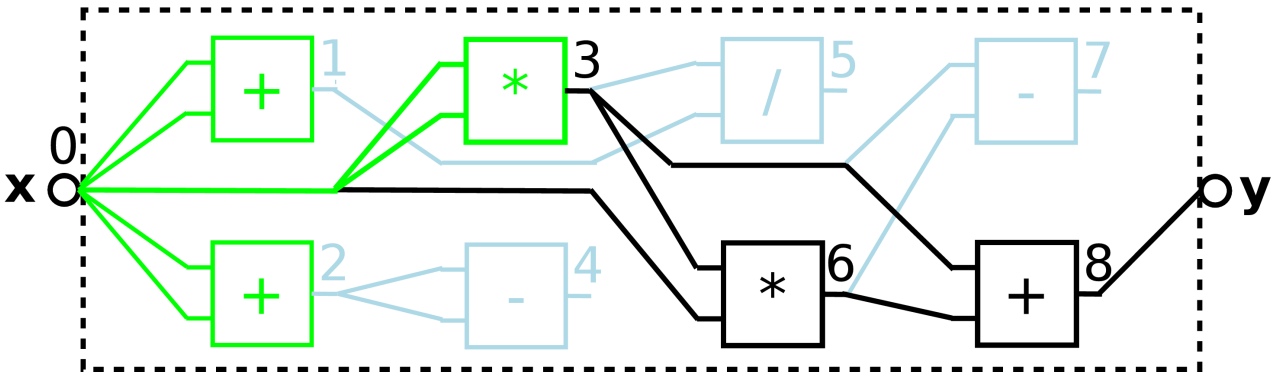
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



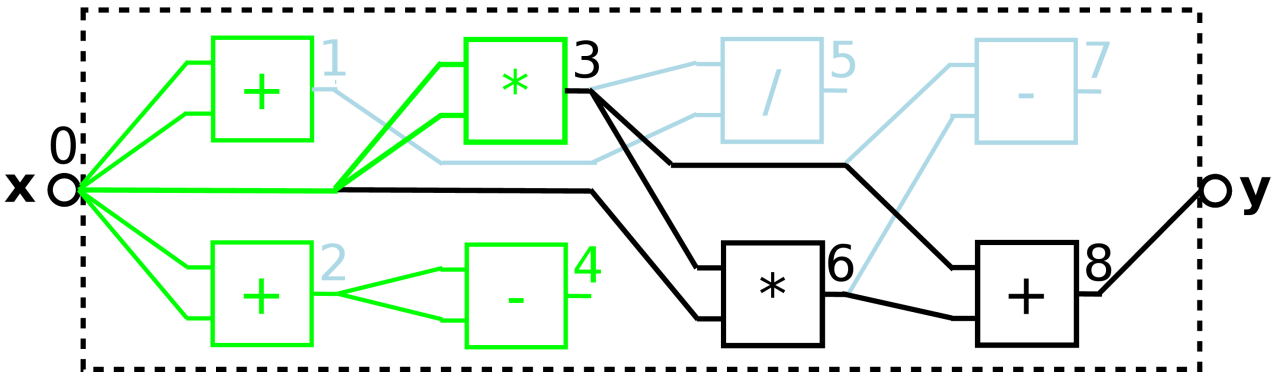
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



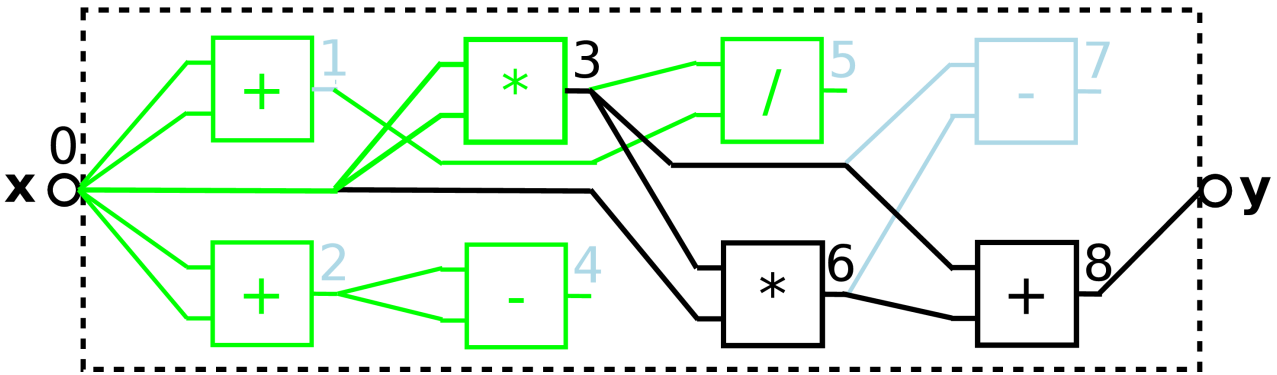
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



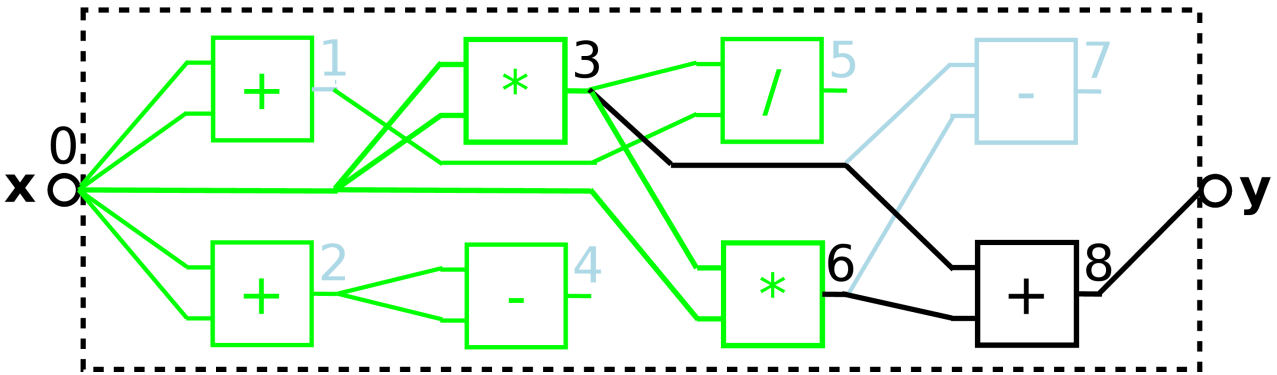
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



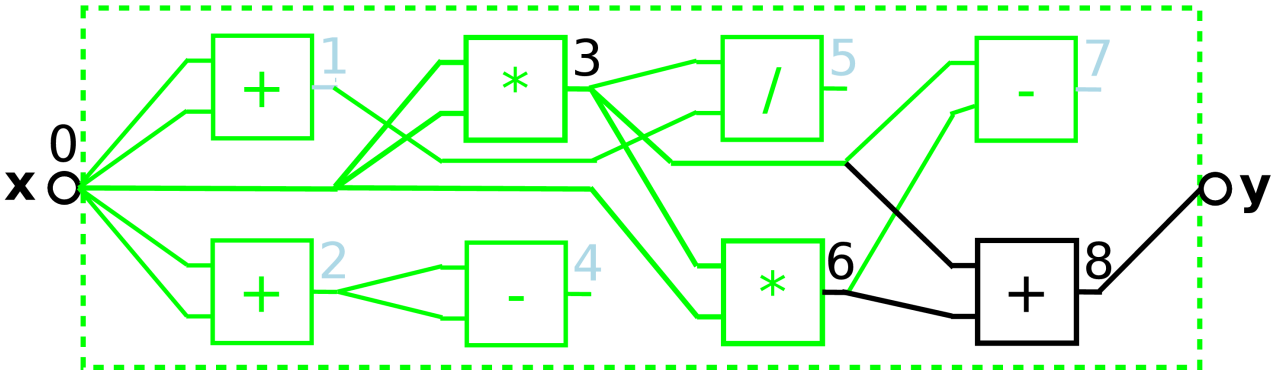
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



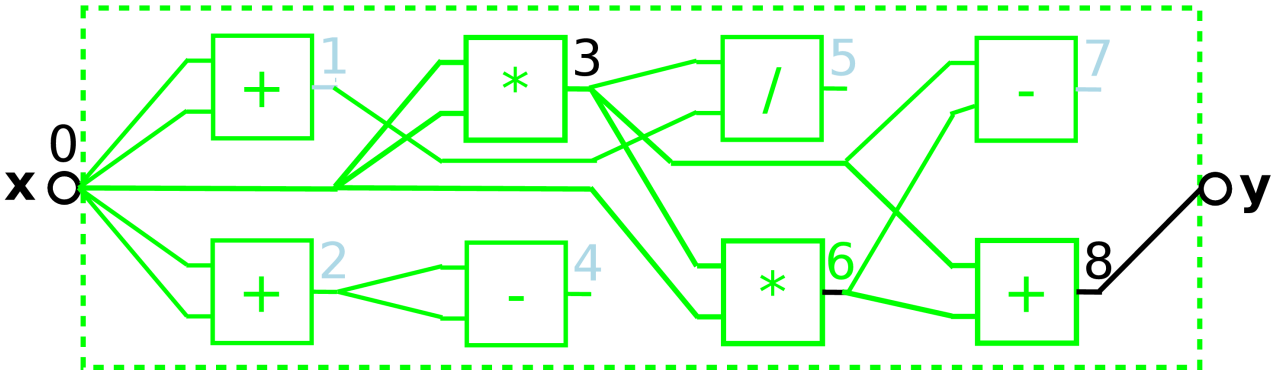
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



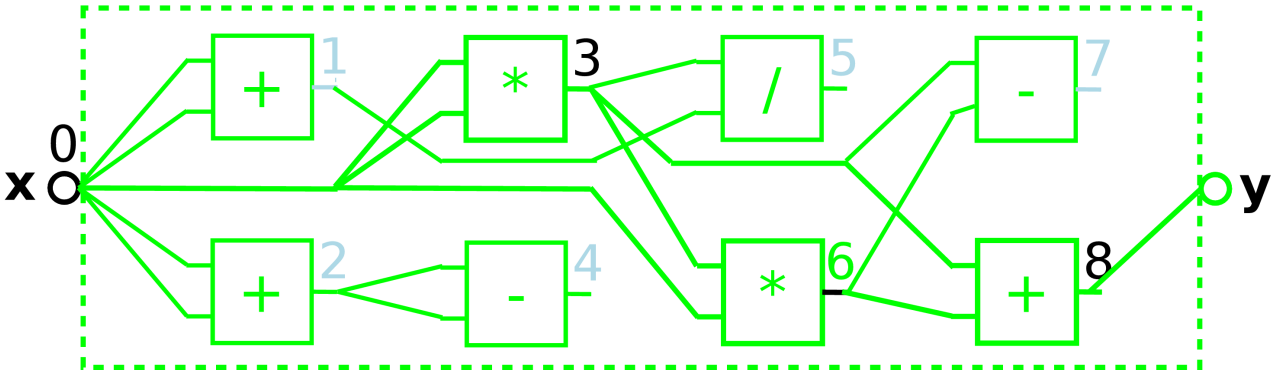
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



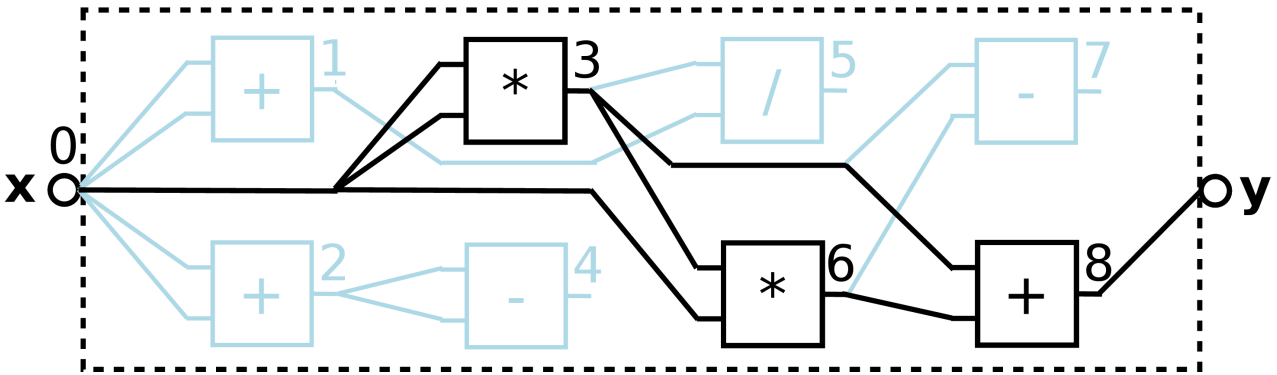
CP – Sequential Interpretation

- Similar to logical circuit evaluation
- + every node is evaluated exactly once
- - inactive nodes are evaluated (Miller has shown, that most efficient percentage of inactive nodes is about 95 %)



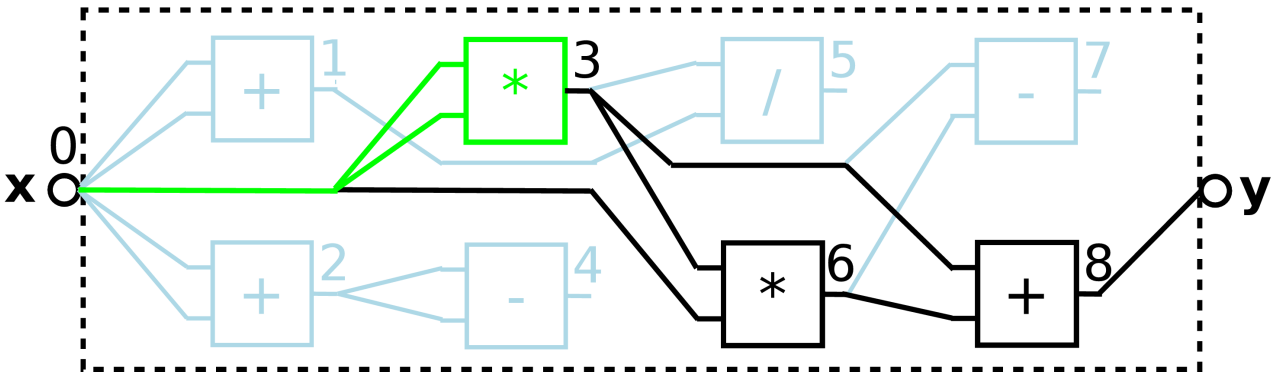
CP – Recursive Descent Active Nodes Selection and Sequential Interpretation

- Active nodes selected using recursive descent
- CP sequentialy interpreted for every element in TDS, but only active nodes
- + every active node is evaluated exactly once
- + inactive nodes are not evaluated
- - additional active nodes selection



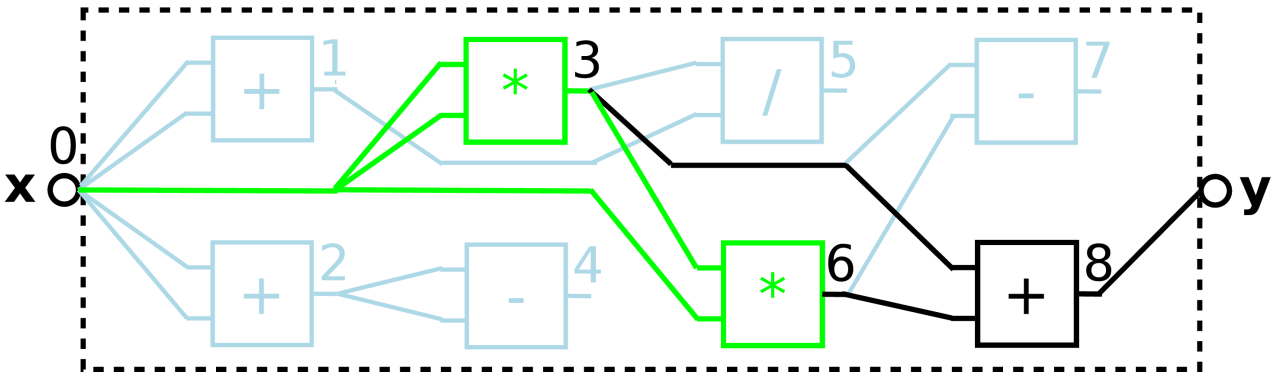
CP – Recursive Descent Active Nodes Selection and Sequential Interpretation

- Active nodes selected using recursive descent
- CP sequentialy interpreted for every element in TDS, but only active nodes
- + every active node is evaluated exactly once
- + inactive nodes are not evaluated
- - additional active nodes selection



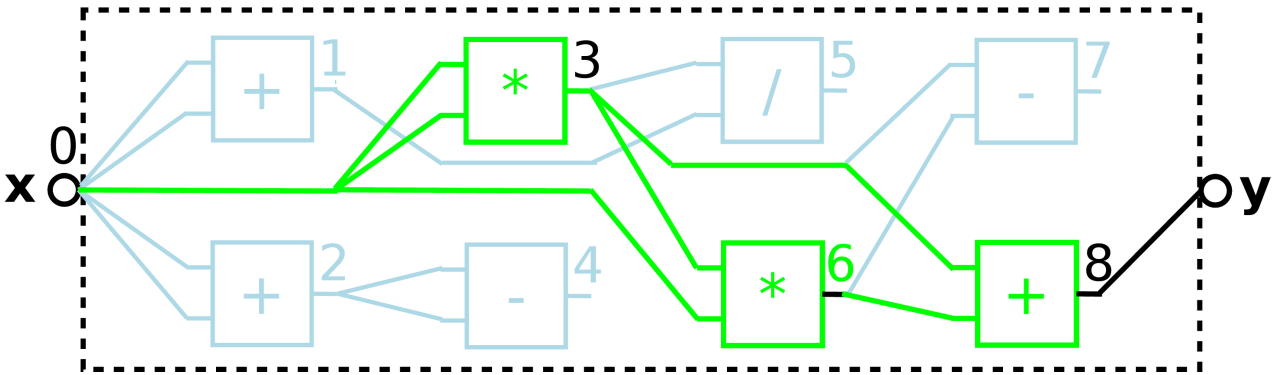
CP – Recursive Descent Active Nodes Selection and Sequential Interpretation

- Active nodes selected using recursive descent
- CP sequentialy interpreted for every element in TDS, but only active nodes
- + every active node is evaluated exactly once
- + inactive nodes are not evaluated
- - additional active nodes selection



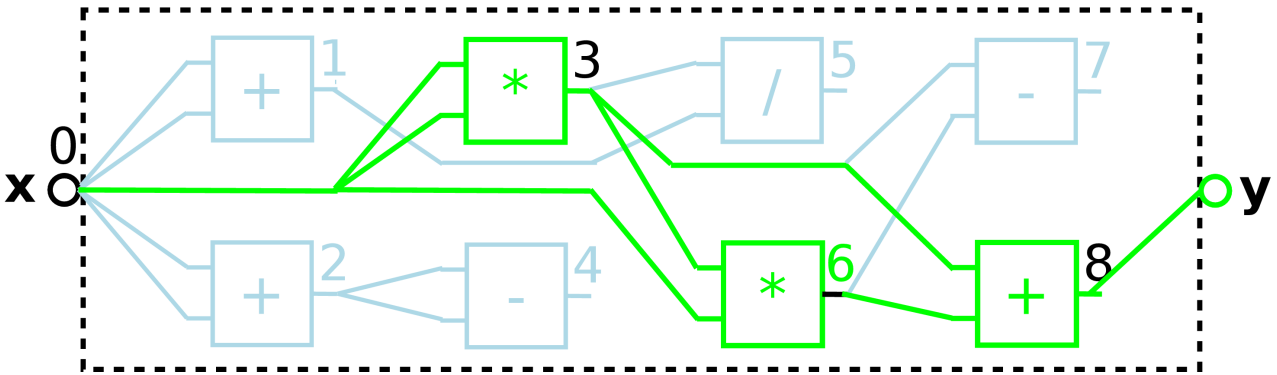
CP – Recursive Descent Active Nodes Selection and Sequential Interpretation

- Active nodes selected using recursive descent
- CP sequentialy interpreted for every element in TDS, but only active nodes
- + every active node is evaluated exactly once
- + inactive nodes are not evaluated
- - additional active nodes selection



CP – Recursive Descent Active Nodes Selection and Sequential Interpretation

- Active nodes selected using recursive descent
- CP sequentialy interpreted for every element in TDS, but only active nodes
- + every active node is evaluated exactly once
- + inactive nodes are not evaluated
- - additional active nodes selection



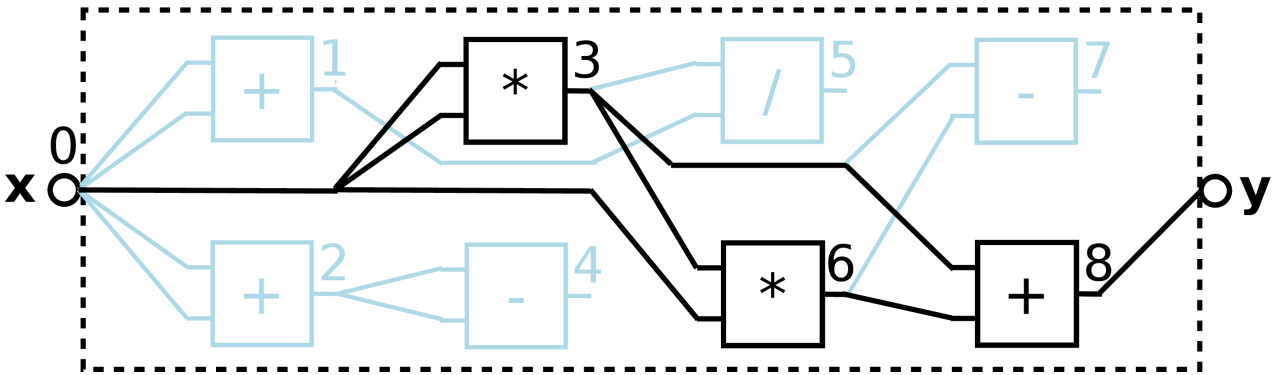
Evolution of CGP genotypes - Mutation

Mutation

- one-point mutation
- an allele at randomly chosen gene is changed to another valid random value
- mutation rate μ_r is a percentage of total number of genes L_g in the genotype. Number of mutations per genotype is then defined

$$\mu_g = \mu_r L_g$$

- μ_r is usually 4 %



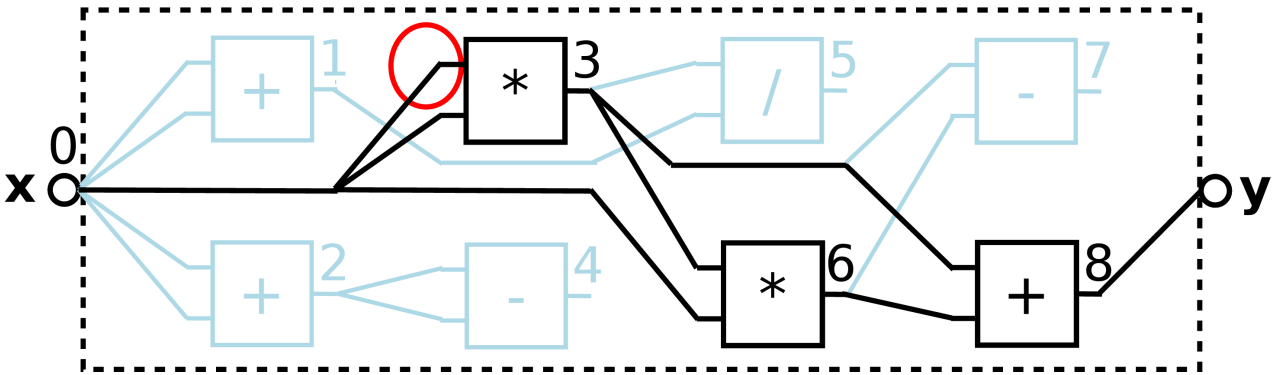
Evolution of CGP genotypes - Mutation

Mutation

- one-point mutation
- an allele at randomly chosen gene is changed to another valid random value
- mutation rate μ_r is a percentage of total number of genes L_g in the genotype. Number of mutations per genotype is then defined

$$\mu_g = \mu_r L_g$$

- μ_r is usually 4 %



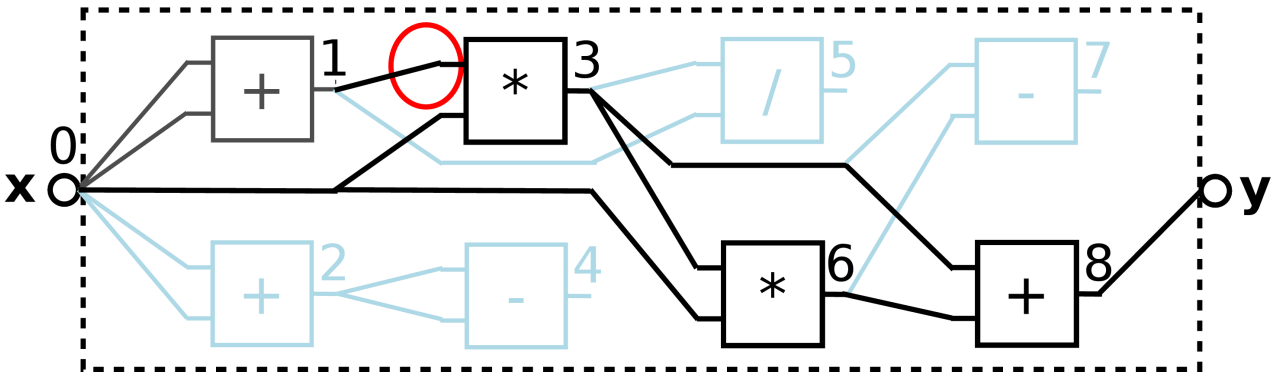
Evolution of CGP genotypes - Mutation

Mutation

- one-point mutation
- an allele at randomly chosen gene is changed to another valid random value
- mutation rate μ_r is a percentage of total number of genes L_g in the genotype. Number of mutations per genotype is then defined

$$\mu_g = \mu_r L_g$$

- μ_r is usually 4 %



Evolution of CGP genotypes

1) Randomly generate CPs in initial population
2) Select the fittest individual (parent) in population
3) Mutate the parent and generate offsprings (next generation)
4) If solution is not found or the generation limit is not reached continue with 2)

References

- Miller, J. F.; Thomson, P.: Cartesian Genetic Programming. In Proc. of the Third European Conference on Genetic Programming (EuroGP2000), LNCS 1820, Springer, 2000: s. 121-132.
- Miller, J.F.: Cartesian Genetic Programming. Springer-Verlag (2011)
- Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk. (2008)

Thank you for your attention