Genetic Programming – basic ideas

These slides were prepared based on chapters 2,3,4 of the book "A Field Guide to Genetic Programming" authored by Riccardo Poli, William Langdon, and Nicholas McPhee and available at http://www.gp-field-guide.org.uk/. The figures shown are from that book.

Fernando Lobo

University of Algarve

GP basic idea

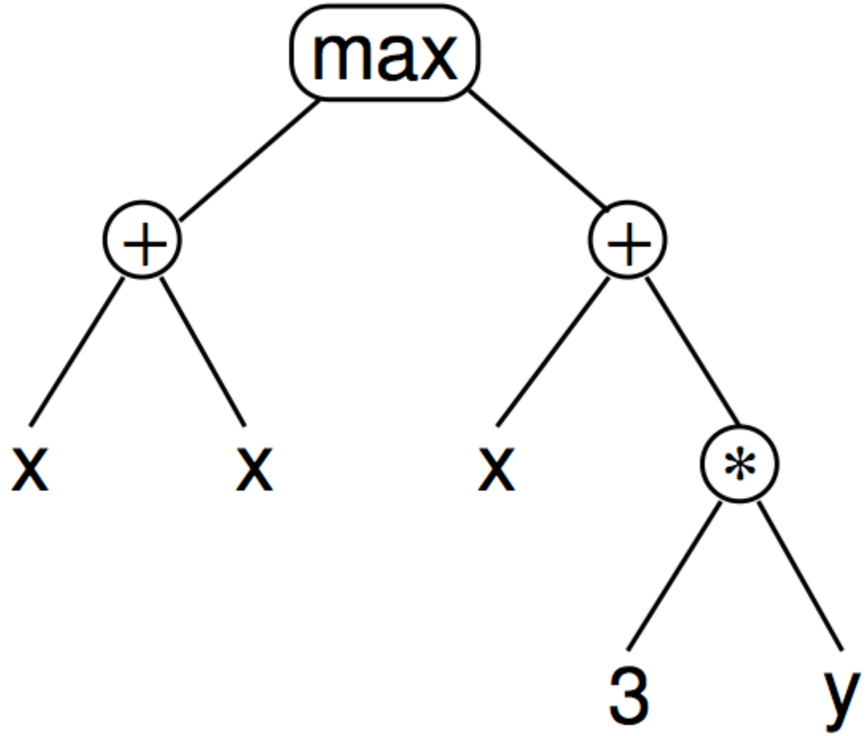
- Goal is automatic programming given a high level description of a problem to solve.
- Structures to be evolved are computer programs.
- GP selects the best programs, and creates new programs by applying variation operators.
- From this perspective, GP is just a special case of a GA.

Representation in Tree-based GP

Syntax trees

- Programs usually represented by syntax trees. Example:

max(x+x,x+3*y)



Terminals and Functions

- Variables and constants (x, y, 3) are leaves in the tree, and are called *terminals*.
- Arithmetic operations (+, *, max) are internal nodes, and are called *functions*.
- *Primitive set* = set of allowed functions and terminals.

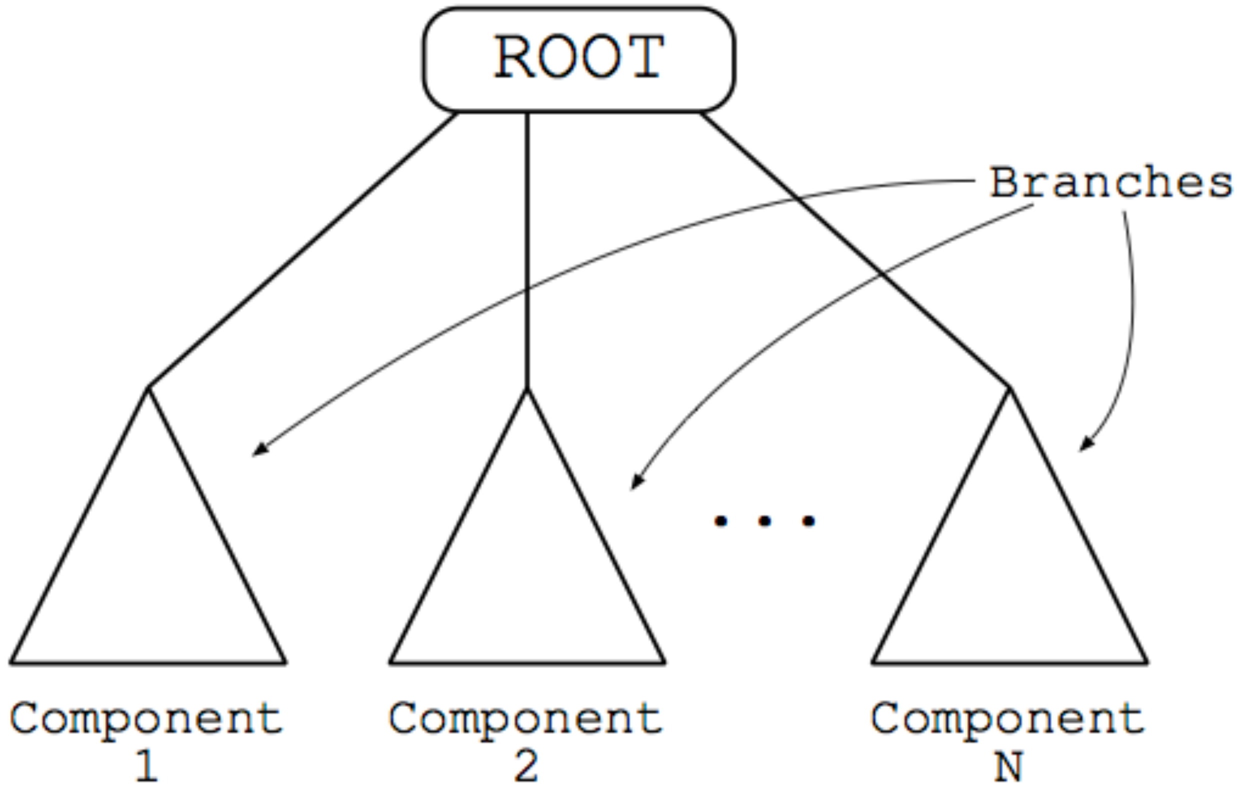
Multiple components

- More advanced forms of GP allow multiple components (such as procedures)

- Representation becomes a set of trees, one for each component.

- Each component is called a *branch*.

Multiple components



Initializing the population

- Typically randomly generated.
- Two major methods:
  - *Full*
  - *Grow*
- Combination of both is often used.

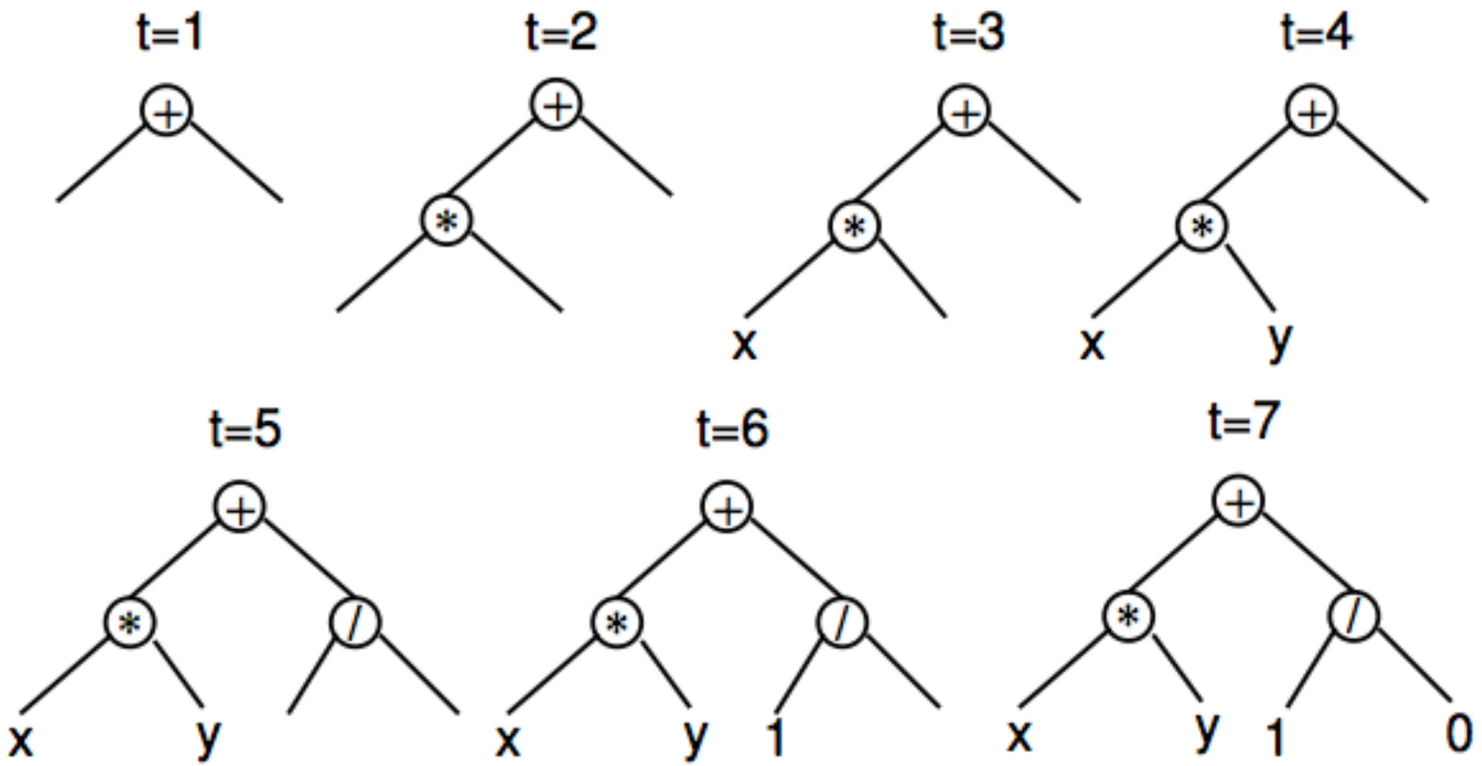
Full and Grow methods

- Individuals generated without exceeding a user specified maximum depth.

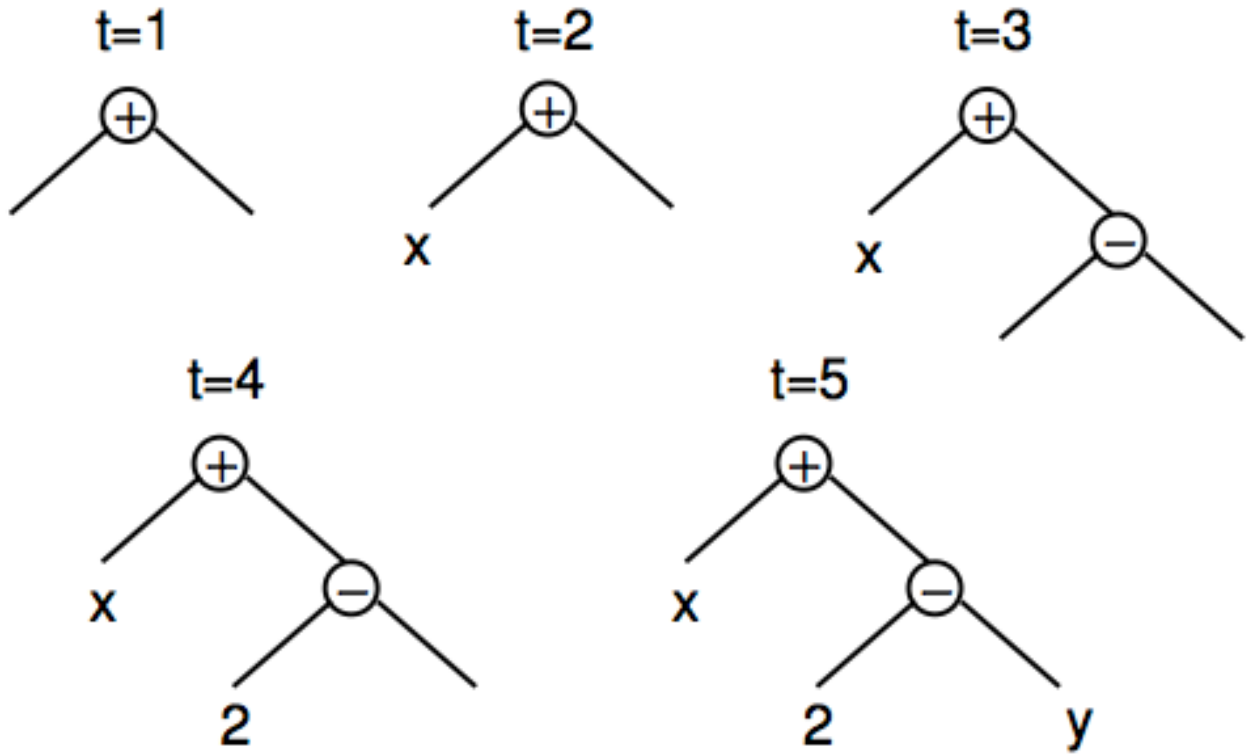
- Full method:
  - Nodes chosen from the function set until the max tree depth is reached. Beyond that only terminals can be chosen.

- Grow method:
  - Nodes are selected from the whole primitive set.

Full method (with max depth = 2)



Grow method (with max depth = 2)



Ramped half-and-half

- Neither full or grow provide trees with a wide variety of sizes and shapes.
- People often used a combination of both, called *ramped half-and-half*.
  - 50% generated with full, 50% with grow.
  - Max depth varied using a range of depth limits (e.g., 2-6).

Selection

- Any selection method used in GAs can be used in GP.
- Most often choice: *tournament selection*.

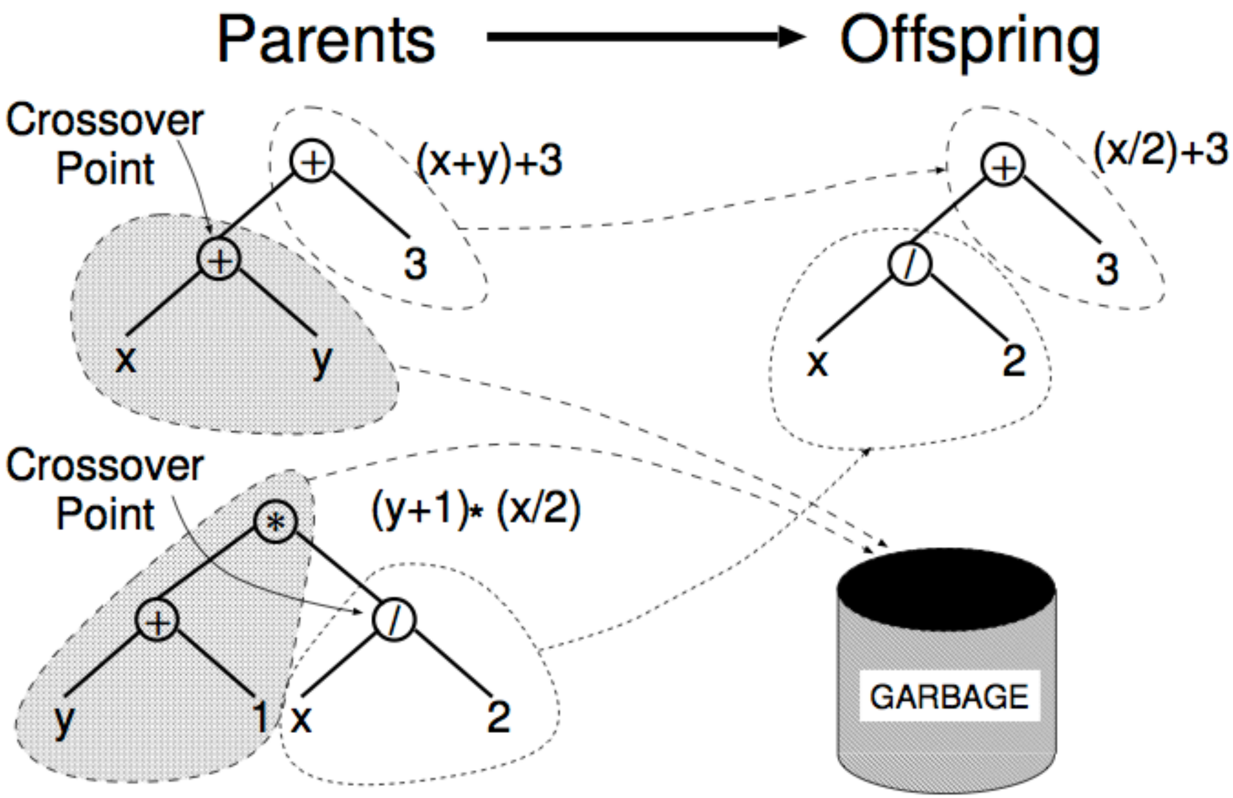
Crossover

- Most common: *subtree crossover*

- Randomly select a crossover point (a node) in each parent tree. Create offspring by replacing subtree rooted at the crossover point in a copy of the first parent with a copy of the subtree rooted at the crossover point in the second parent.

- Crossover points not selected with the same probability. Why?
  - typical choice: functions 90%, terminals 10%

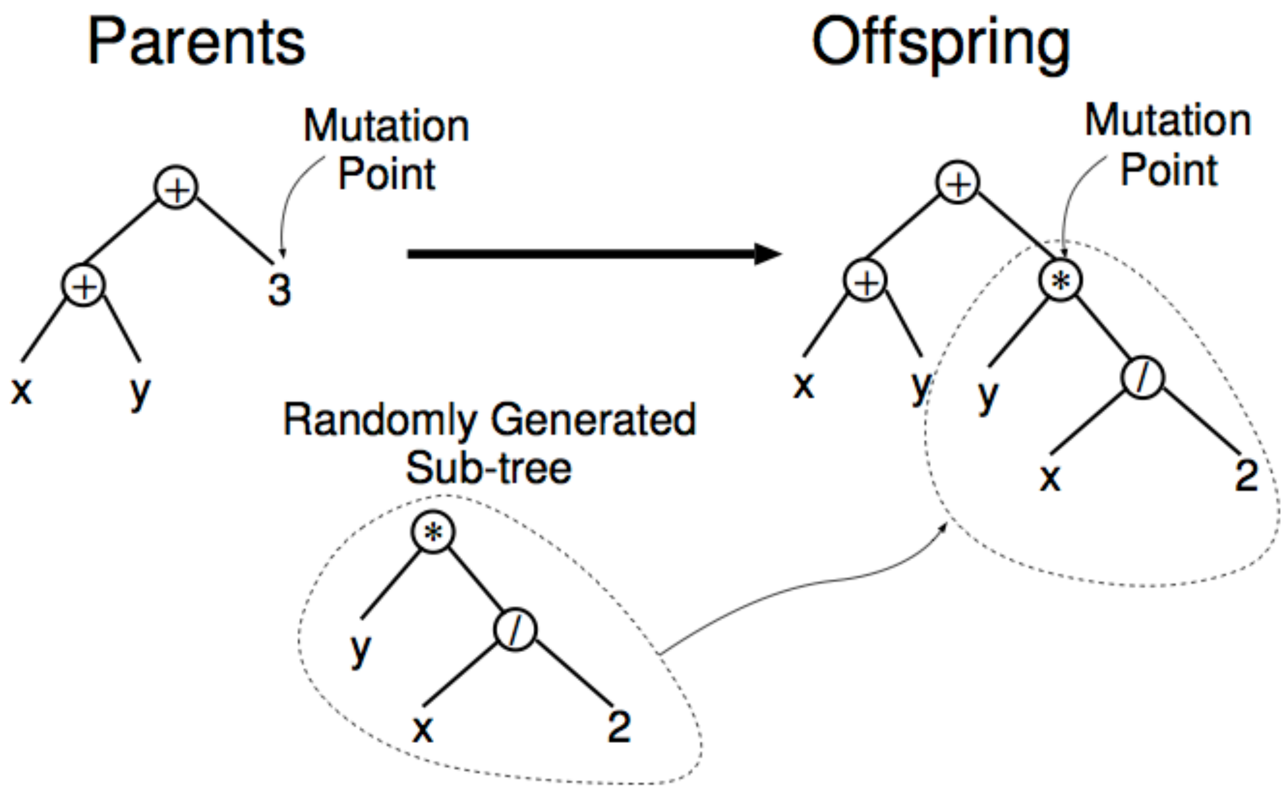
Subtree crossover



Mutation

- Most common: *subtree mutation*
- Randomly select a mutation point (a node) in a tree and substitute the subtree rooted there with a randomly generated subtree.

Subtree mutation



Point mutation

- Another type is *point mutation*
  - Randomly select a node and substitute its value with another primitive of the same arity.

- Usually applied on a per-node basis (like in bit-flip mutation in GAs)

- Subtree mutation only applied once per individual.

Operator rates

- GAs: crossover + mutation
- GP: either crossover or mutation or reproduction.
- Choice is probabilistic based on operator rates. Typical values:
  - crossover rate around 90%
  - mutation rate around 1%
  - reproduction rate = whatever is left to 100%

Preparatory steps for applying GP

- What is the terminal set?
- What is the function set?
- What is the fitness measure?
- Parameters for controlling the run.
- Stop criterion and result of the run.

Terminal set

- Programs to be evolved are constrained to a domain-specific language.
- The function and terminal sets define such a language (i.e., the kind of programs that can be generated).

Terminal set consists of

- External inputs, usually variables (ex: x,y)
- Function with no arguments (ex: rand(), move-right())
  - May have side effects (e.g., control a robot)
- Constants
  - Use a terminal that represents an *ephemeral random constant*.
  - Every time it is chosen, a different random constant is generated (but remains fixed for the entire run).

Function set

- Depends on the nature of the problem.
- Can be as simple as the arithmetic functions (+ - * /).
- Can use typical programming constructs.
- Can use specialized commands (e.g., move-forward, turn-left, turn-right).

Examples of primitives in GP

Function Set	
Kind of Primitive	*Example(s)*
Arithmetic	+, *, /
Mathematical	sin, cos, exp
Boolean	AND, OR, NOT
Conditional	IF-THEN-ELSE
Looping	FOR, REPEAT
⋮	⋮

Terminal Set	
Kind of Primitive	*Example(s)*
Variables	x, y
Constant values	3, 0.45
0-arity functions	rand, go_left

Closure

- Type consistency
  - Functions must return values of the same type.
  - Conversions are possible.
- Evaluation safety
  - Functions cannot fail at run time.
  - Use protected versions of operators (e.g., dividing by 0 gives 1).

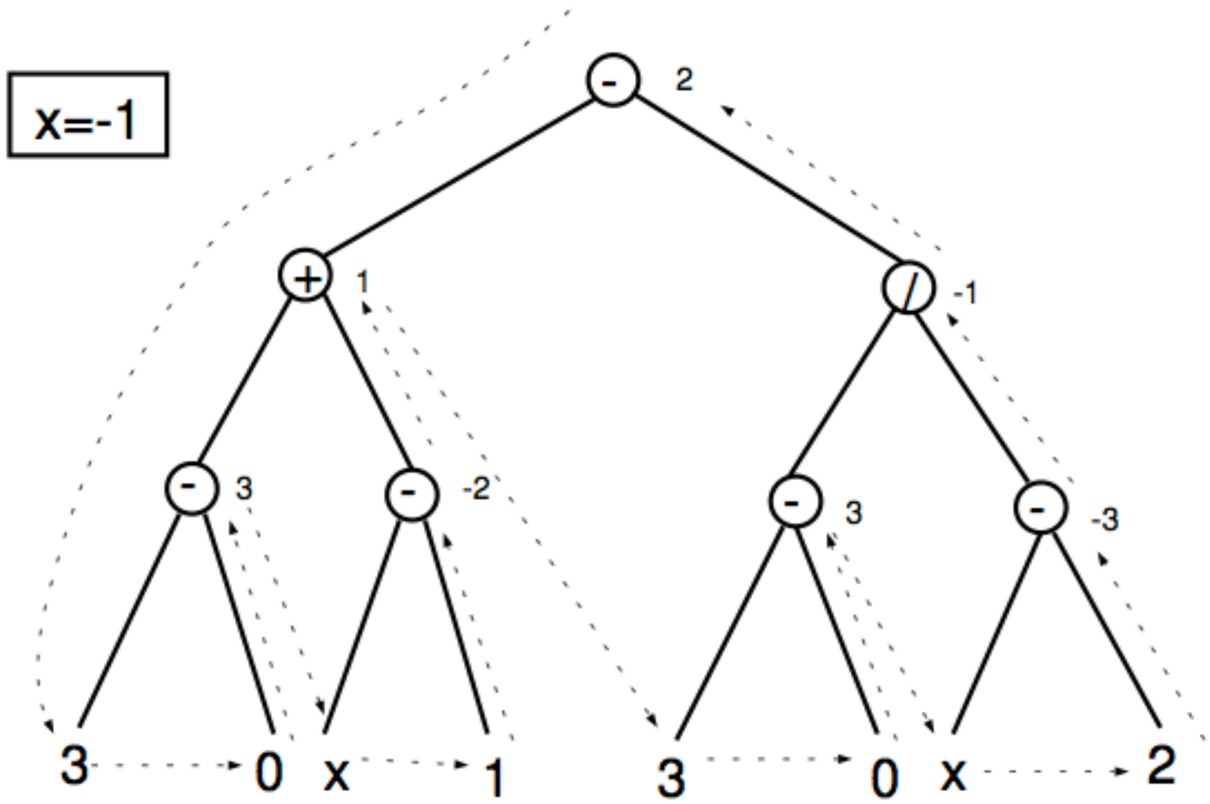
Fitness function

- How to measure the fitness of a computer program?
- Answer: based on how well that program solves the given problem.

Fitness evaluation

- Requires executing all the programs in the population
  - most common solution is to use an interpreter
- Often multiple executions are needed.
- Can be time consuming.

Interpretation of a syntax tree



Fitness cases

- Fitness of a program typically depends on the results of its execution on many different inputs.

- These different inputs are called *fitness cases*.

Typical GP parameters

- Large populations (thousands or more).
- Initialization using ramped half-and-half with a depth range 2-6.
- Large operator rate for crossover, around 90%.
- 50-50 mix of crossover and a variety of mutation operators also popular. In this case, smaller populations can also work well.

Termination

- Like in GAs, a max number of generations or a problem-specific success predicate.

- The *best-so-far* individual is designated the result of the run.

Example of a GP run

- Goal: Automatically create a computer program that for a given value of x, outputs the value of the quadratic polynomial $x^2 + x + 1$ in the range [-1.0,+1.0].

- Also called symbolic regression.

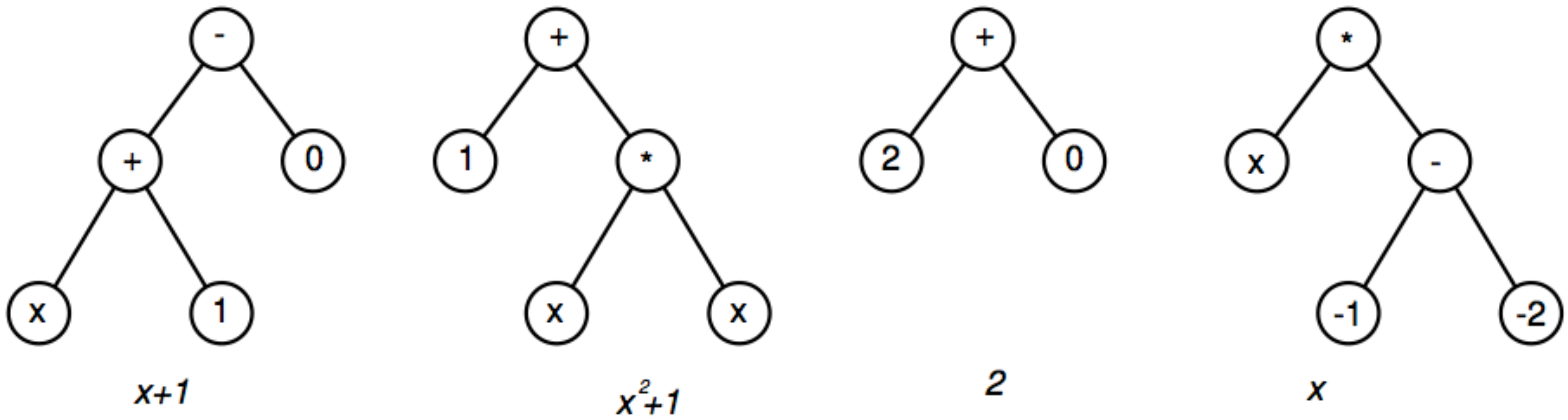
Preparatory steps

- Terminal set: $T = \{x, \Re\}.$
- Function set: $F = \{+, -, *, \%\},$
- $\Re$ stands for ephemeral random constants, say in [-5.0,+5.0]

Parameters

Objective:	Find program whose output matches $x^2 + x + 1$ over the range $-1 \leq x \leq +1$.
Function set:	$+$, $-$, % (protected division), and $\times$; all operating on floats
Terminal set:	x, and constants chosen randomly between -5 and $+5$
Fitness:	sum of absolute errors for $x \in \{-1.0, -0.9, \ldots 0.9, 1.0\}$
Selection:	fitness proportionate (roulette wheel) non elitist
Initial pop:	ramped half-and-half (depth 1 to 2. 50% of terminals are constants)
Parameters:	population size 4, 50% subtree crossover, 25% reproduction, 25% subtree mutation, no tree size limits
Termination:	Individual with fitness better than 0.1 found

Initial population

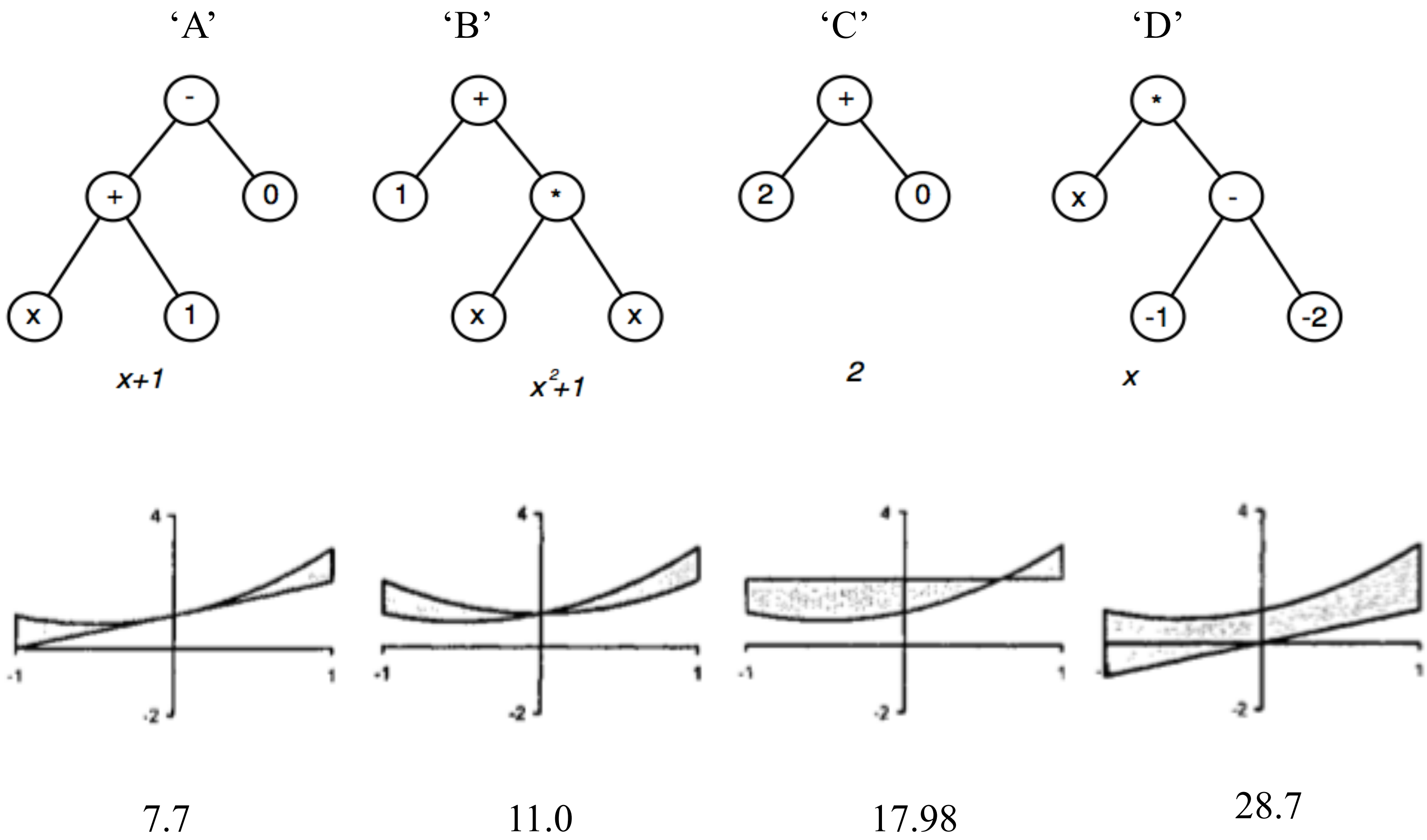


Fitness cases

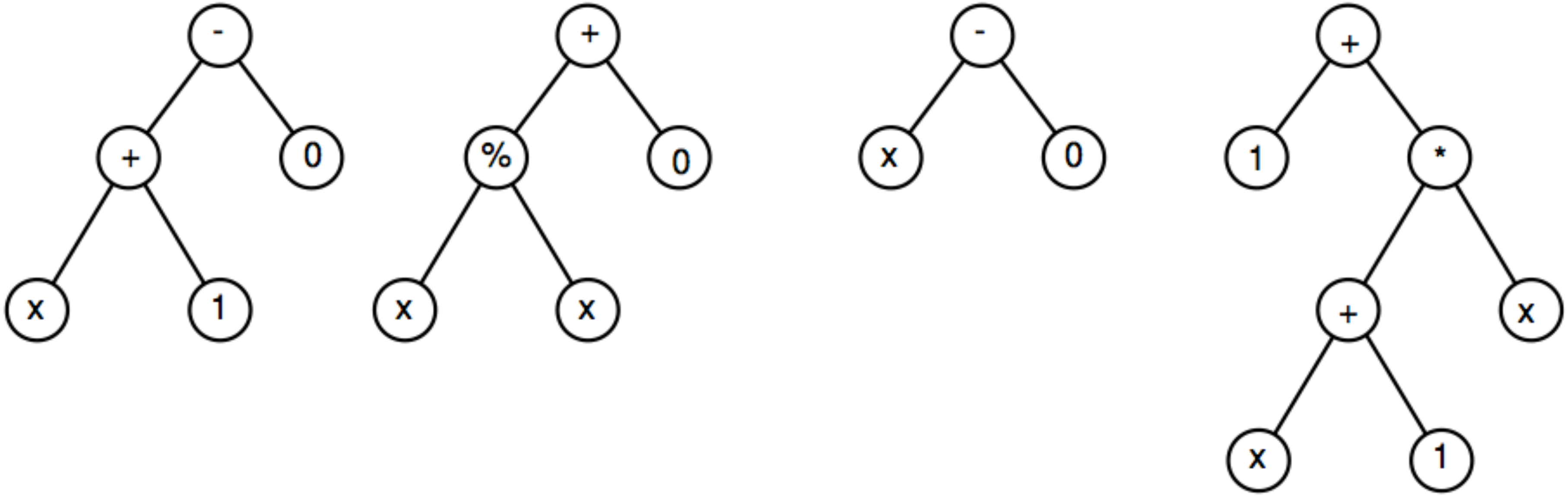
Fitness will be the sum of absolute errors over the fitness cases.

Independent variable (Input) *X*	Dependent Variable (Output) *Y*
-1.0	1.00
-0.9	0.91
-0.8	0.84
-0.7	0.79
-0.6	0.76
-0.5	0.75
-0.4	0.76
-0.3	0.79
-0.2	0.84
-0.1	0.91
0	1.00
0.1	1.11
0.2	1.24
0.3	1.39
0.4	1.56
0.5	1.75
0.6	1.96
0.7	2.19
0.8	2.44
0.9	2.71
1.0	3.00

Fitness evaluation



After one generation



$x+1$

Reproduction (copy of ‘A’)

1

Mutation of ‘C’ on node ‘2’.

x

Crossover of ‘A’ and ‘D’, with node ‘+’ selected on ‘A’ and leftmost ‘*x*’ selected on ‘D’.

$x^2 + x + 1$

Crossover of ‘B’ and ‘A’, with leftmost ‘*x*’ selected on ‘B’ and ‘+’ selected on ‘A’.

What to know more about GP?

- Read the book by Poli, Langdon, and McPhee.