Recap:
Learning by Evolution of a Species

Start with a set of problems.
Start with a population of rules.

For some number of iterations:

a. For each rule,

- Run the rule against each problem. Collect the results of each run.

- Combine the results to determine the **fitness** of the rule.

b. Construct a new population of rules from the current set based on the fitness of each rule.

The effect: a preference for successful phenotypes.

How to Generate New Rules: Direct Reproduction

Direct reproduction equates to asexual reproduction in the animal kingdom. A single parent rule gives rise to a child:

A rule: `1#00# –> 0111#`

Generalization: `1##0# –> 0111#`

A rule: `1#00# –> 0111#`

Specialization: `1#00# –> 01110`

Specialization: `1100# –> 0111#`

Consider our rules for the wall-following robot:

- `if x1 and not x2 then move east`
- `if x2 and not x3 then move south`
- `if x3 and not x4 then move west`
- `if x4 and not x1 then move north`
- `default action: move north`

How to Generate New Rules: Crossover

Crossover equates to sexual reproduction in the animal kingdom. Two (or more) parents rule gives rise to a child:

Parent 1:	`1#00# –> 0111#`
Parent 2:	`01#10 –> 11001`
Child 1:	`1#00# –> 11001`
Child 2:	`1#010 –> 11001`
Child 3:	`1##10 –> 1101#`

Consider our rules for the wall-following robot:

- `if x1 and not x2 then move east`
- `if x2 and not x3 then move south`
- `if x3 and not x4 then move west`
- `if x4 and not x1 then move north`
- `default action: move north`

How to Generate New Rules: Mutation

Mutation equates to mutation in the animal kingdom! Sometimes, a child differs in a seemingly random way from its intended genotype:

Rule:	`1#00# –> 0111#`
Mutation!	`1#01# –> 0111#`

Rule:	`1#00# –> 0111#`
Mutation!	`1#00# –> 0011#`

What distinguishes mutation from direct reproduction is that, in direct reproduction, a change is made to the parent rule for a **reason**. Mutation happens.

Consider our rules for the wall-following robot:

- `if x1 and not x2 then move east`
- `if x2 and not x3 then move south`
- `if x3 and not x4 then move west`
- `if x4 and not x1 then move north`
- `default action: move north`

Exercise:
Fixing the Learning Parameters

How often should rules reproduce?

How often should rules mutate?

How many new rules should be created?

How strong should the new rules be?

The Result: Genetic Algorithms

The practical results:

- a preference for successful patterns
  (genotypes that correspond to successful phenotypes)
- self-optimizing systems
- focused search

Another nice advantage:

- The GA idea makes no cognitive claims.

A Variation: Genetic Programming

Rules are like programs; why not apply the idea of genetic evolution directly to programs?

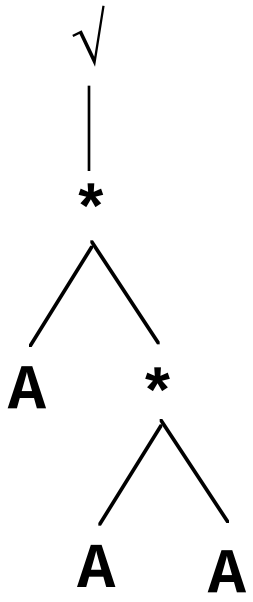
- broaden the alphabet to functions and terminals, where terminals are constants and variables
- broaden from one "link" to many

As a simple example, consider a program to compute the orbital period P of a planet, given the planet's average distance from the sun A.

If we express P and A in units relative to Earth's values, then the formula is:

$$P = \sqrt{A^3}$$

Suppose that we wished to learn this formula, based on empirical data gathered from sky gazing?

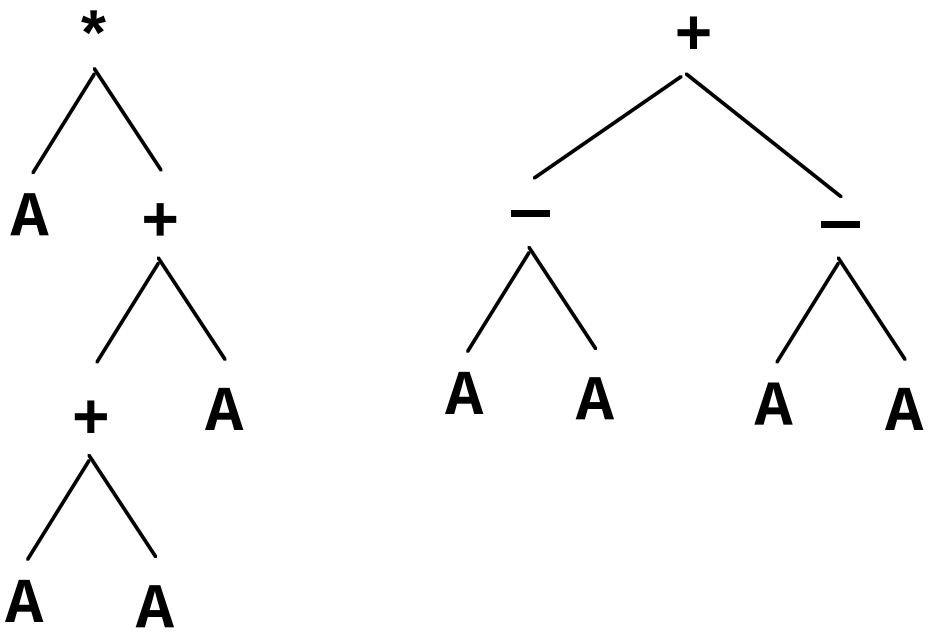


Writing a Program via GP (1)

Genetic programming works much like what we have done:

1. Identify the set of functions and terminals that you will allow in the program.

2. Generate an initial population of (perhaps random) programs created out of the alphabet.



Writing a Program via GP (2)

3. Calculate the fitness of each program in the population by running it on a set of training cases.

PLANET	A	Actual P
Venus	0.72	0.61
Earth	1.00	1.00
Mars	1.52	1.84
Jupiter	5.20	11.90
Saturn	9.53	29.40
Uranus	19.10	83.50

Writing a Program via GP (3)

4. Apply genetic operators to the population to create a new population. Use a program's fitness as its strength when choosing programs to reproduce and replace.