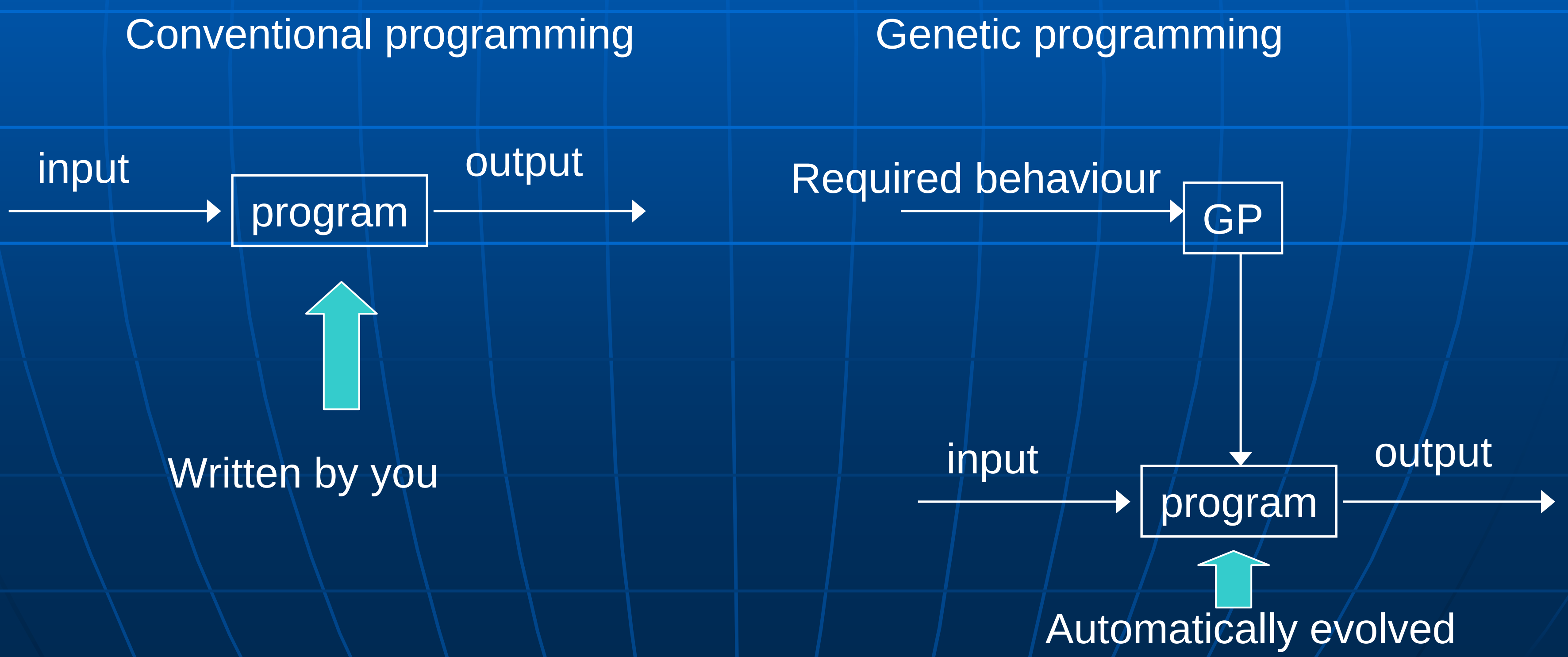
ECM3412/ECMM409 Overview of Genetic Programming

(Adapted/used slides from Martin Henz, John Koza, and Jason Lohn)

General Idea

- Overview of an Evolutionary Algorithm to evolve programs



Background

- Origin of GP: Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.
- Work since early 1980s; series of books (4 volumes) in various editions
- Considered a branch of AI

Algorithm outline

1. Generate random "programs"
2. Evaluate programs using training data
3. Selectively modify population of programs using cross-over and mutation etc.
4. If a good program is found, finish, else go to 2

i.e. this is just an EA, but where we are evolving programs.

E.g. way in the future

Evolve an operating system, or word processor, etc ...

Fitness: users or simulated users work with it for an amount of time, and rate its behaviour.

It's not so crazy – initial population could be seeded with previously human-developed systems. Evolution could be constrained to bits of it (e.g. develop fast code for search-and-replace,etc...)

Back to 21st Century

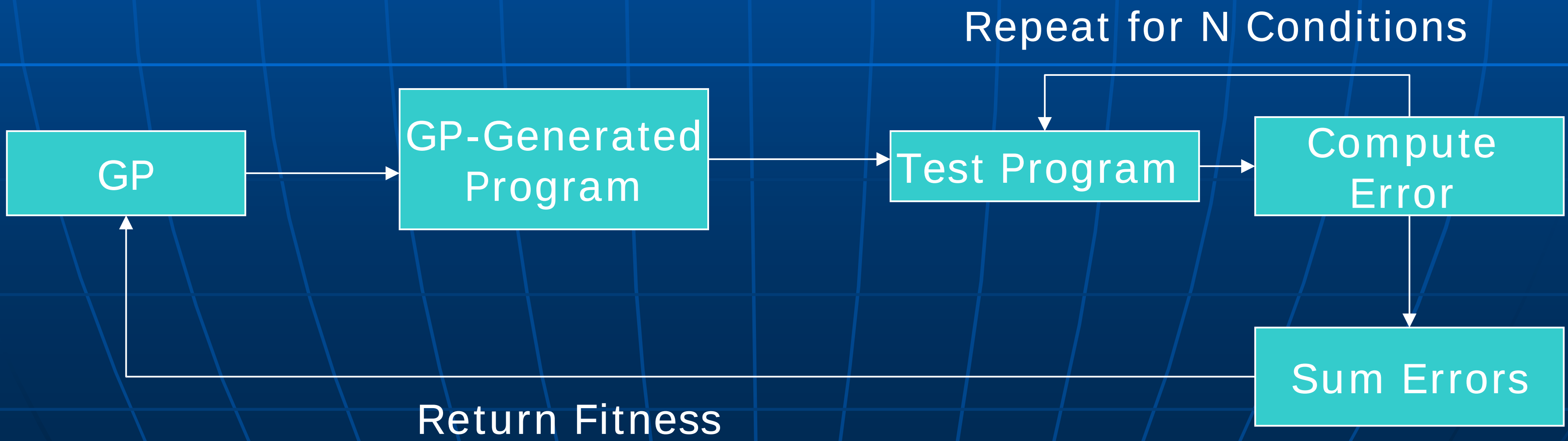
Current GP is like this:

Programming tasks:

- navigation code for a mobile robot
- curve fitting
- antenna design
- circuit design
- prediction, etc...
- They tend to be standard things you might expect EAs to be applied to, but some of which need to be represented as programs.

Fitness Evaluation

- The fitness evaluation is more complex than with standard EAs
- `Normal' EA: a chromosome represents a design, a schedule, etc. We just evaluate it and give it a score.
- GP: the chromosome is a program. To evaluate it, we have to run it, and test its behaviour over a range of potential inputs.



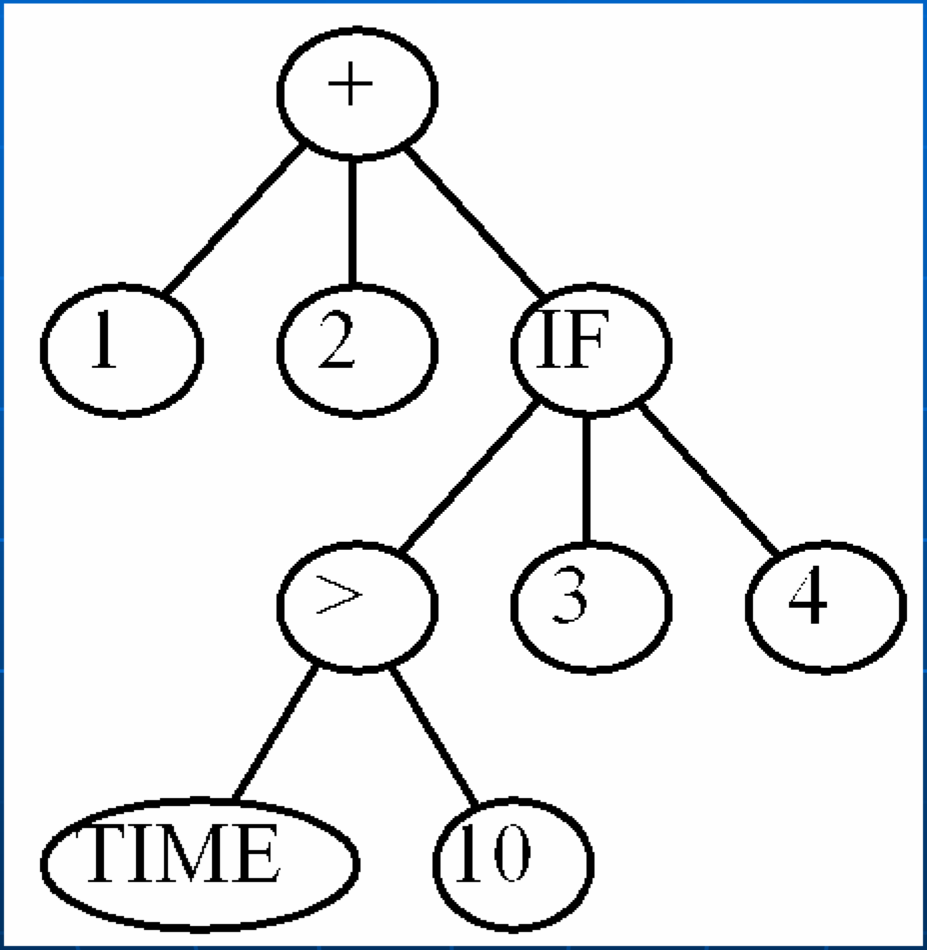
A COMPUTER PROGRAM IN C

```
int foo (int time)
{
    int temp1, temp2;
    if (time > 10)
        temp1 = 3;
    else
        temp1 = 4;
    temp2 = temp1 + 1 + 2;
    return (temp2);
}
```

OUTPUT OF C PROGRAM

Time	Output
0	6
1	6
2	6
3	6
4	6
5	6
6	6
7	6
8	6
9	6
10	6
11	7
12	7

Same program as a tree




```
(+ 1 2 (IF (> TIME 10) 3 4))
```

CREATING RANDOM PROGRAMS

We need a:

Function Set: e.g. PLUS, MINUS, TIMES, DIV, IF

Terminal Set: e.g. X, Y, <any real number>

Associated syntax rules:

e.g. in this case, PLUS can have only two children from F and/or T. MINUS can have only two children from F and/or T. IF has 4 children, A, B and C, where the meaning is “IF A > B then return C else return D”.

Then we can create a random program with an algorithm like this

```
Maximum depth:  5;
Definition:
Start: choose a random member of F for the root;  set its depth = 1
Repeat:
     Choose a function node X which does not have its children yet;
     If (X < Maximum depth – 1) Randomly choose appropriate children for that
       node, and set the childrens’  depth to D +1, where D is the depth of their parent.
     If (X = Maximum depth – 1) Randomly choose appropriate children for that node,
       but ensuring that they are all terminals.
```

Example random program creation

Start with a randomly chosen function node

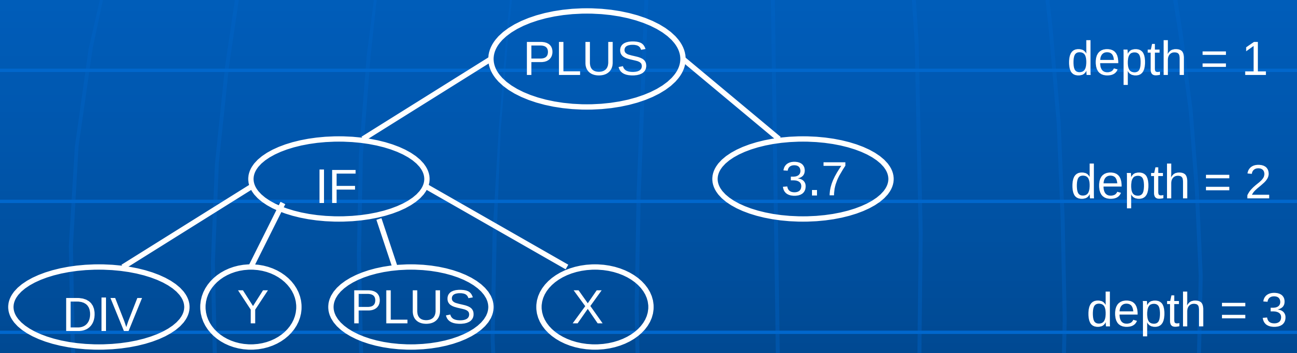
depth = 1

Example random program creation

Now choose a random childless function node and generate appropriate random children

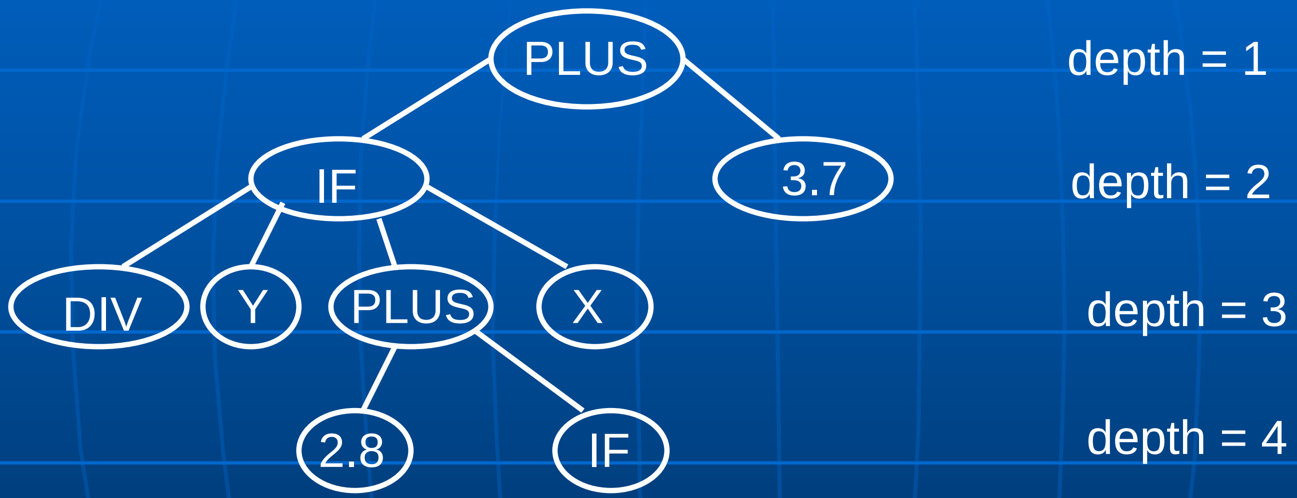
Example random program creation

Again choose a random childless function node and generate appropriate random children



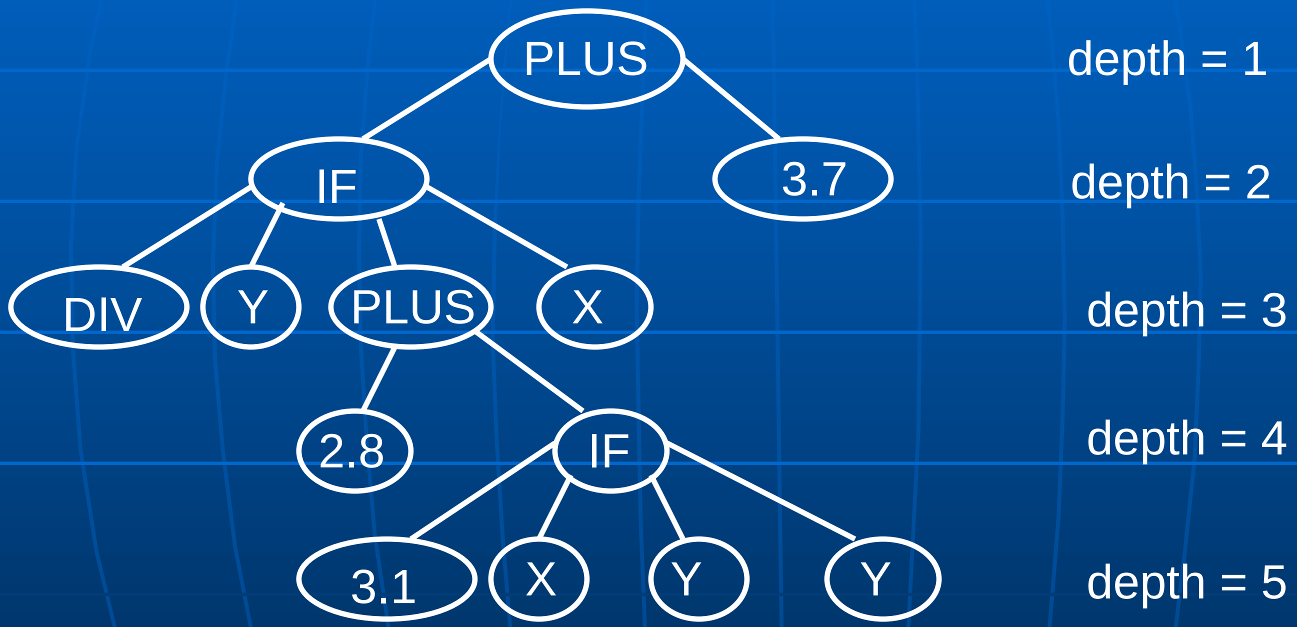
Example random program creation

And again …



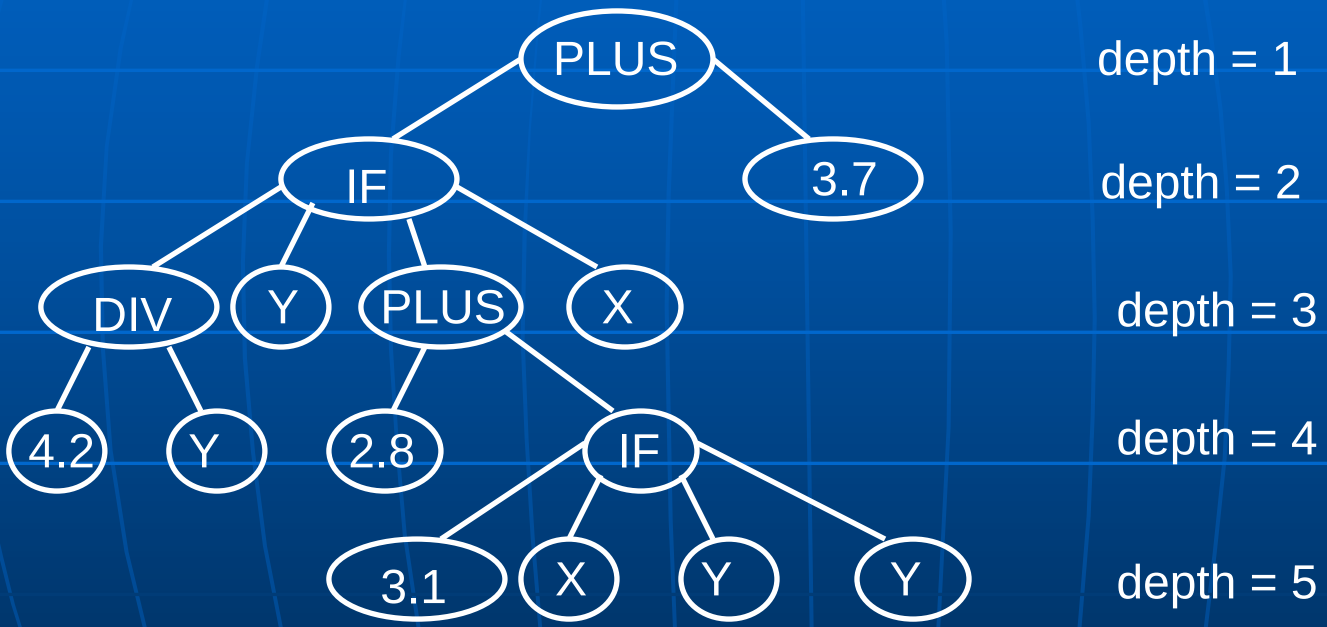
Example random program creation

And again … this time we expanded a function with depth Max -1, so children must be Terminals

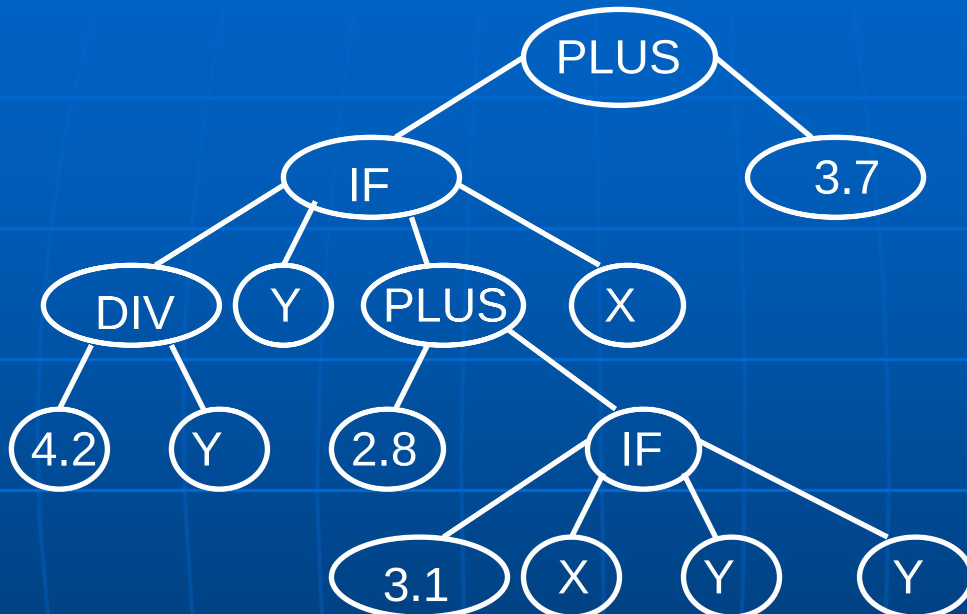


Example random program creation

And again … and now there are no Function nodes remaining without children



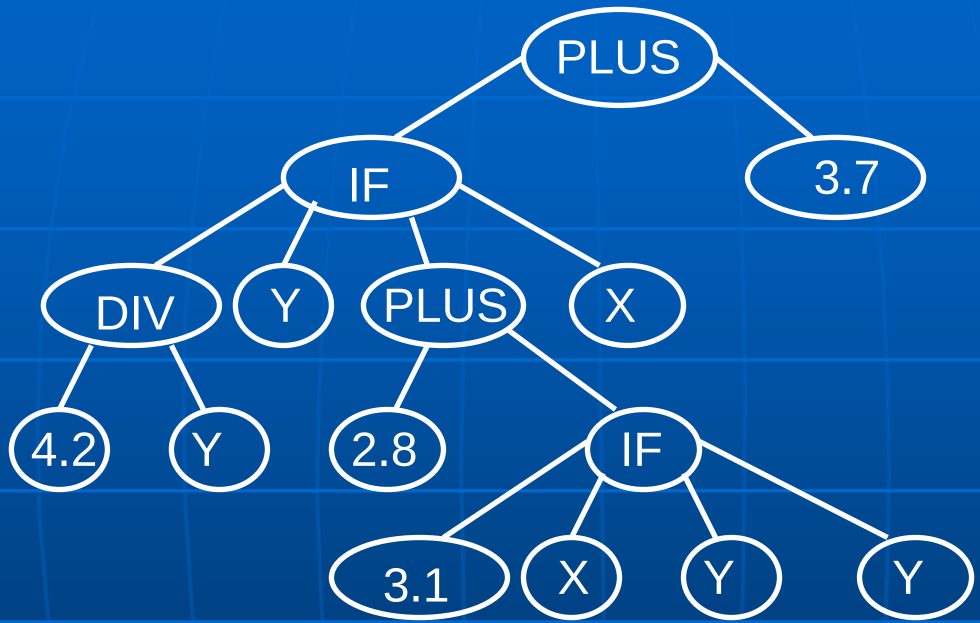
So, what have we got?



Interpreting this in a natural way, this is the following program

```
Input (X, Y)
If X < 3.1, then tmp1 = Y, else tmp1 = Y
tmp2 = tmp1 + 2.8
tmp3 = 4.2/Y
If tmp3 > Y, then tmp4 = tmp2 else tmp4 = X
tmp5 = tmp4 + 3.7
output:  tmp5
```

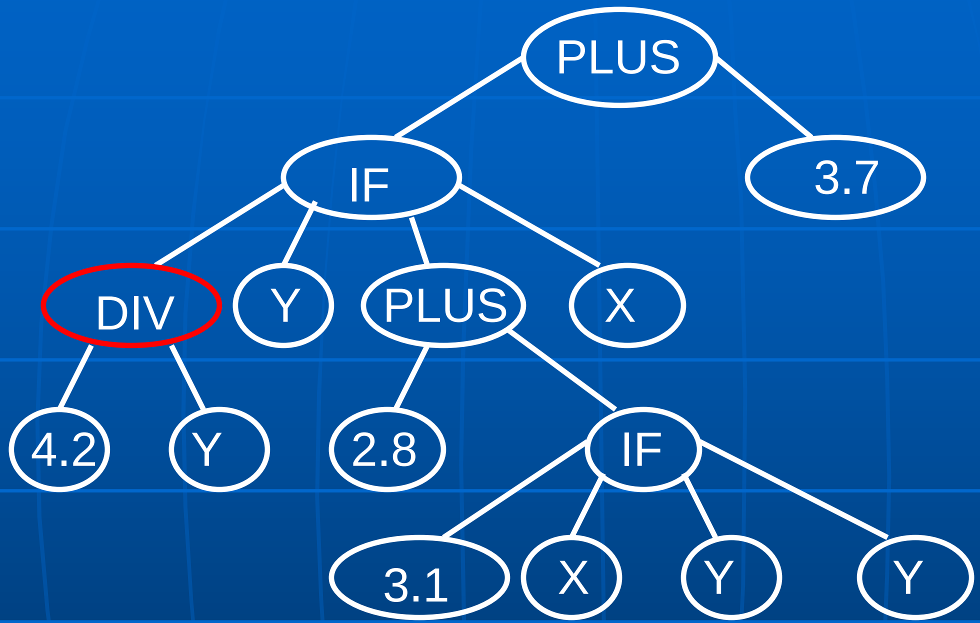
Mutation



Standard approach is to choose a subtree at random, remove it, and then generate a new subtree in its place. This is usually biased towards nodes with high depth.

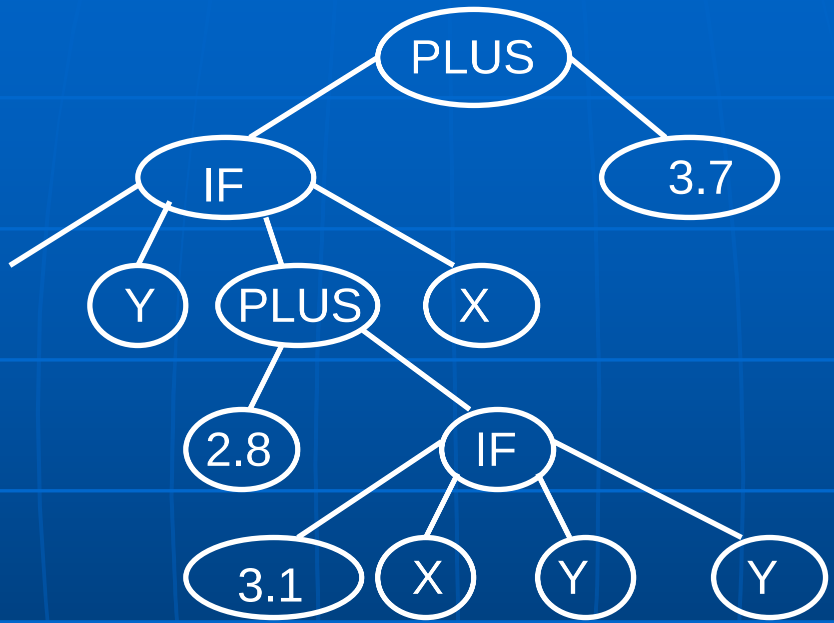
Choosing a subtree is equivalent to choosing a node.
For example:

Mutation



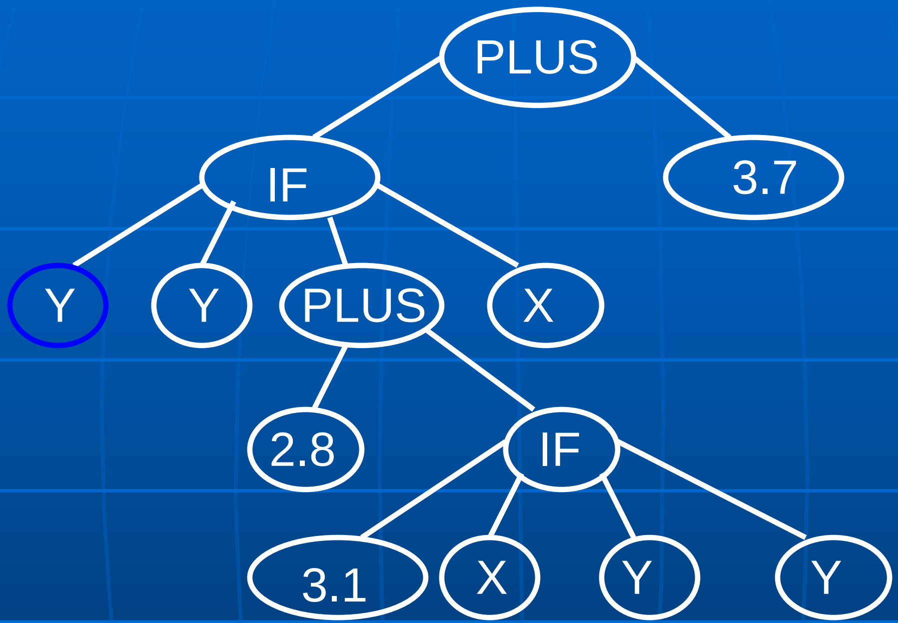
Random choice of node

Mutation



Remove the subtree rooted at that node

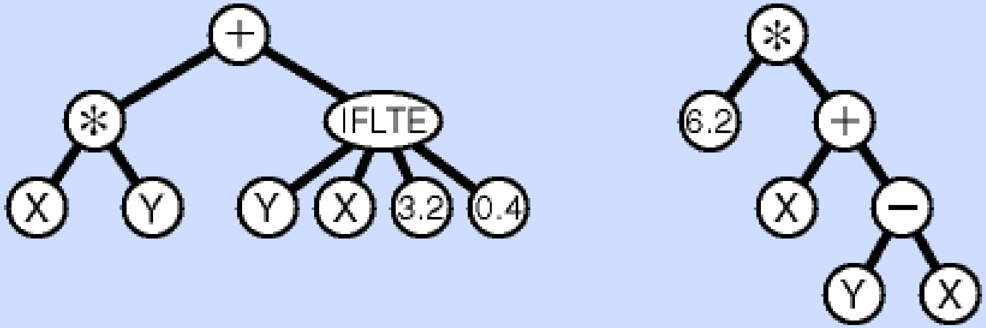
Mutation



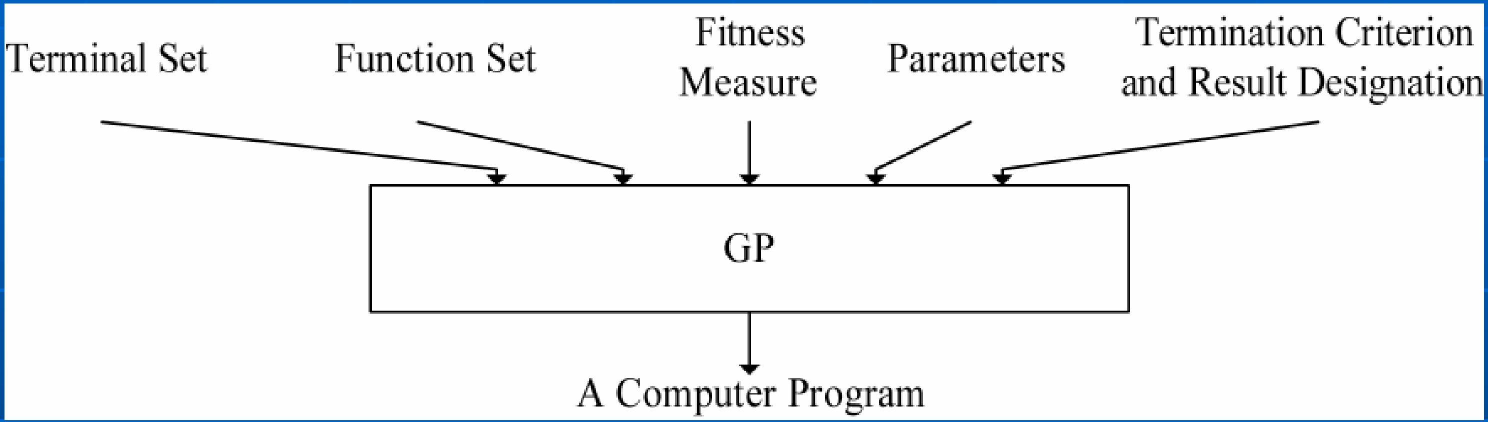
Generate a new subtree at that node, following the usual rules about depth etc. In this case the new subtree happens to be a randomly chosen terminal, but it could have been entire tree going down to level 5.

What has this mutation done to the program?

Crossover



FIVE MAJOR PREPARATORY STEPS FOR GP



- Determining the set of terminals
- Determining the set of functions
- Determining the fitness measure
- Determining the parameters for the run
- Determining the method for designating a result and the criterion for terminating a run

ILLUSTRATIVE GP RUN

SYMBOLIC REGRESSION

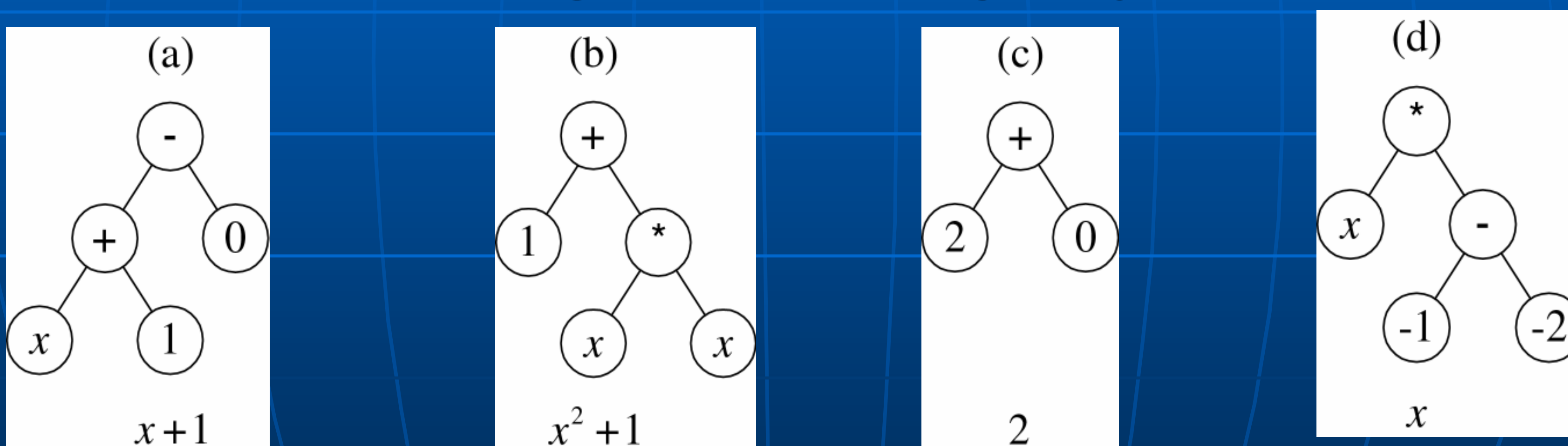
Independen t variable *X*	Dependent variable *Y*
-1.00	1.00
-0.80	0.84
-0.60	0.76
-0.40	0.76
-0.20	0.84
0.00	1.00
0.20	1.24
0.40	1.56
0.60	1.96
0.80	2.44
1.00	3.00

PREPARATORY STEPS

	Objective:	**Find a computer program with one input (independent variable X) whose output equals the given data**
1	**Terminal set:**	**T = {X, Random-Constants}**
2	**Function set:**	**F = {+, -, *, %}**
3	**Fitness:**	**The sum of the absolute value of the differences between the candidate program's output and the given data (computed over numerous values of the independent variable *x* from –1.0 to +1.0)**
4	**Parameters:**	**Population size *M* = 4**
5	**Termination:**	**An individual emerges whose sum of absolute errors is less than 0.1**

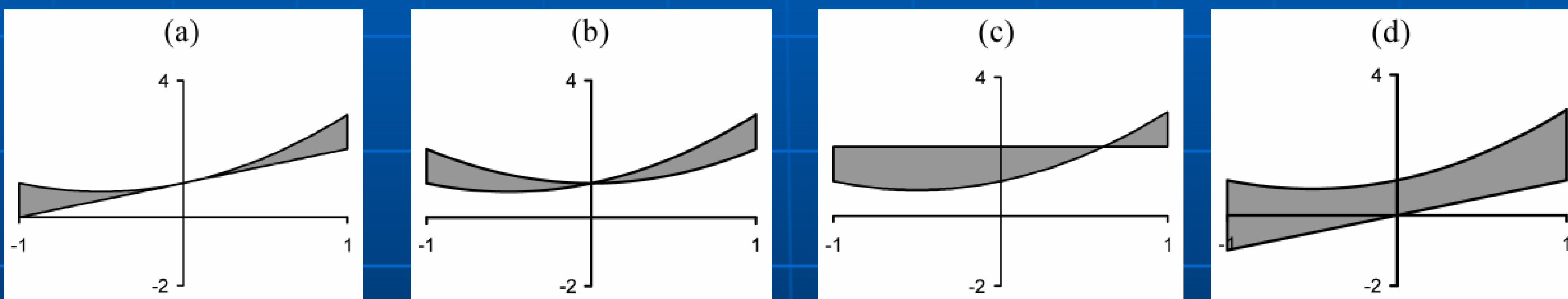
SYMBOLIC REGRESSION

POPULATION OF 4 RANDOMLY CREATED INDIVIDUALS FOR GENERATION 0



SYMBOLIC REGRESSION $x^2 + x + 1$

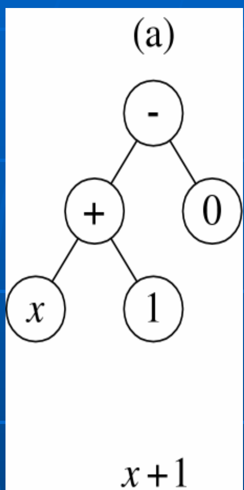
FITNESS OF THE 4 INDIVIDUALS IN GEN 0



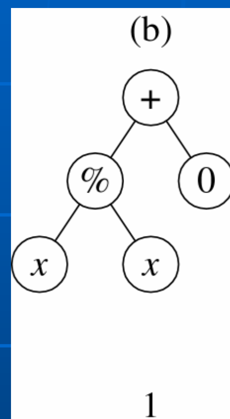
$x + 1$	$x^2 + 1$	2	x
0.67	**1.00**	**1.70**	**2.67**

SYMBOLIC REGRESSION $x^2 + x + 1$

GENERATION 1

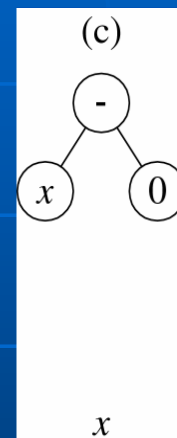


Copy of (a)

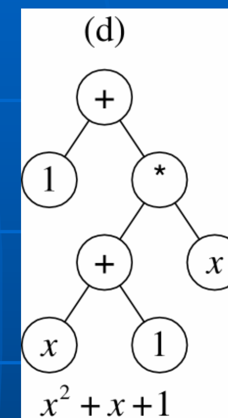


Mutant of (c)

picking “2” as mutation point

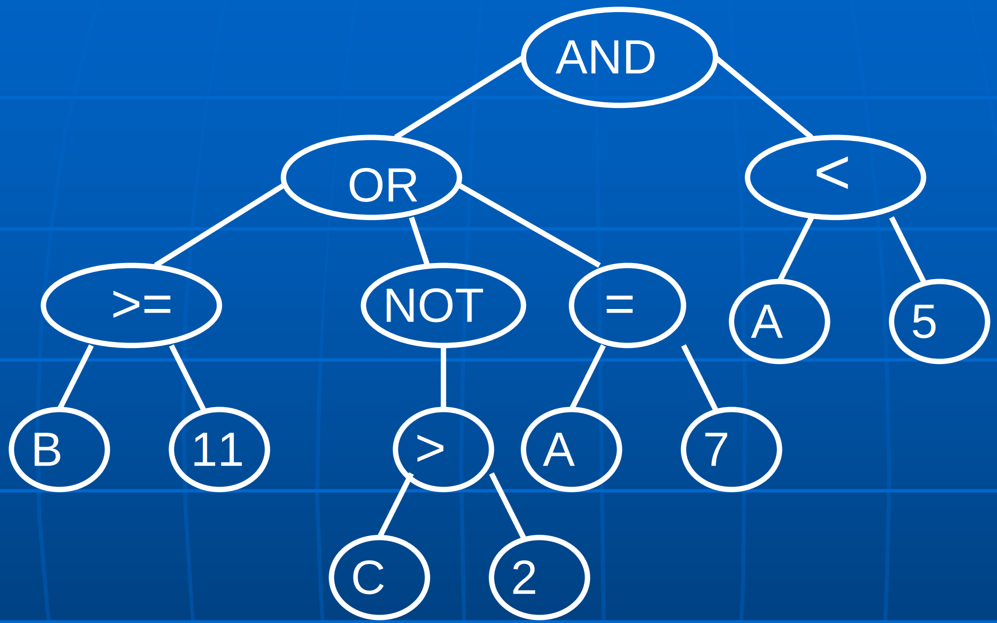


First offspring of crossover of (a) and (b) picking “+” of parent (a) and left-most “x” of parent (b) as crossover points



Second offspring of crossover of (a) and (b) picking “+” of parent (a) and left-most “x” of parent (b) as crossover points

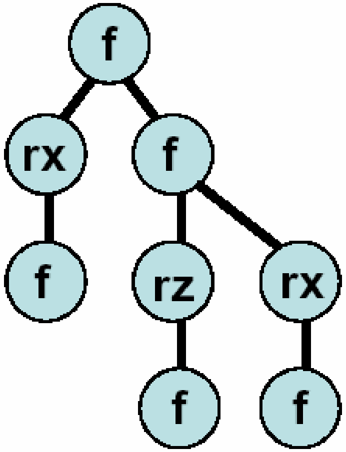
Different Function/Terminal Sets

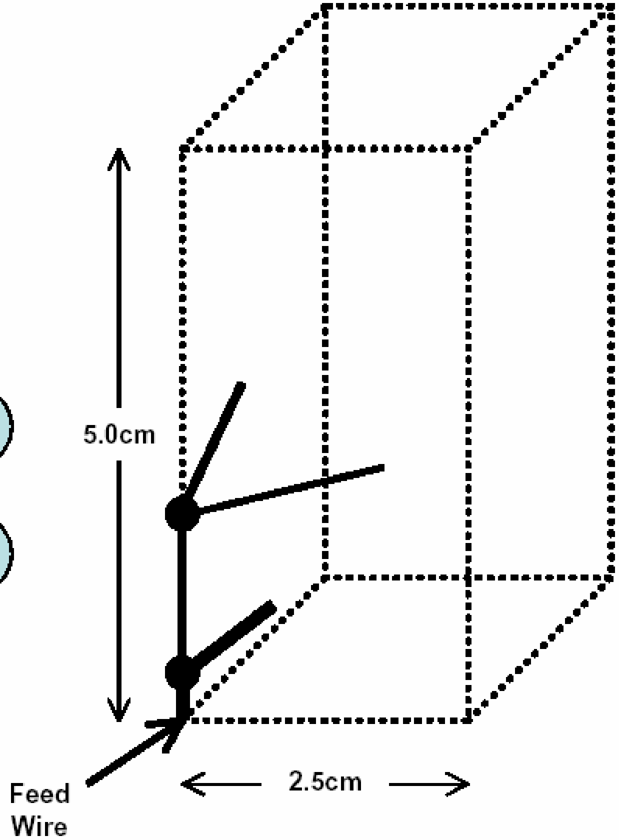


Programs which compute logical functions of the data;
Very useful in data mining – e.g. this could be medical data concerning levels of certain proteins in blood test results

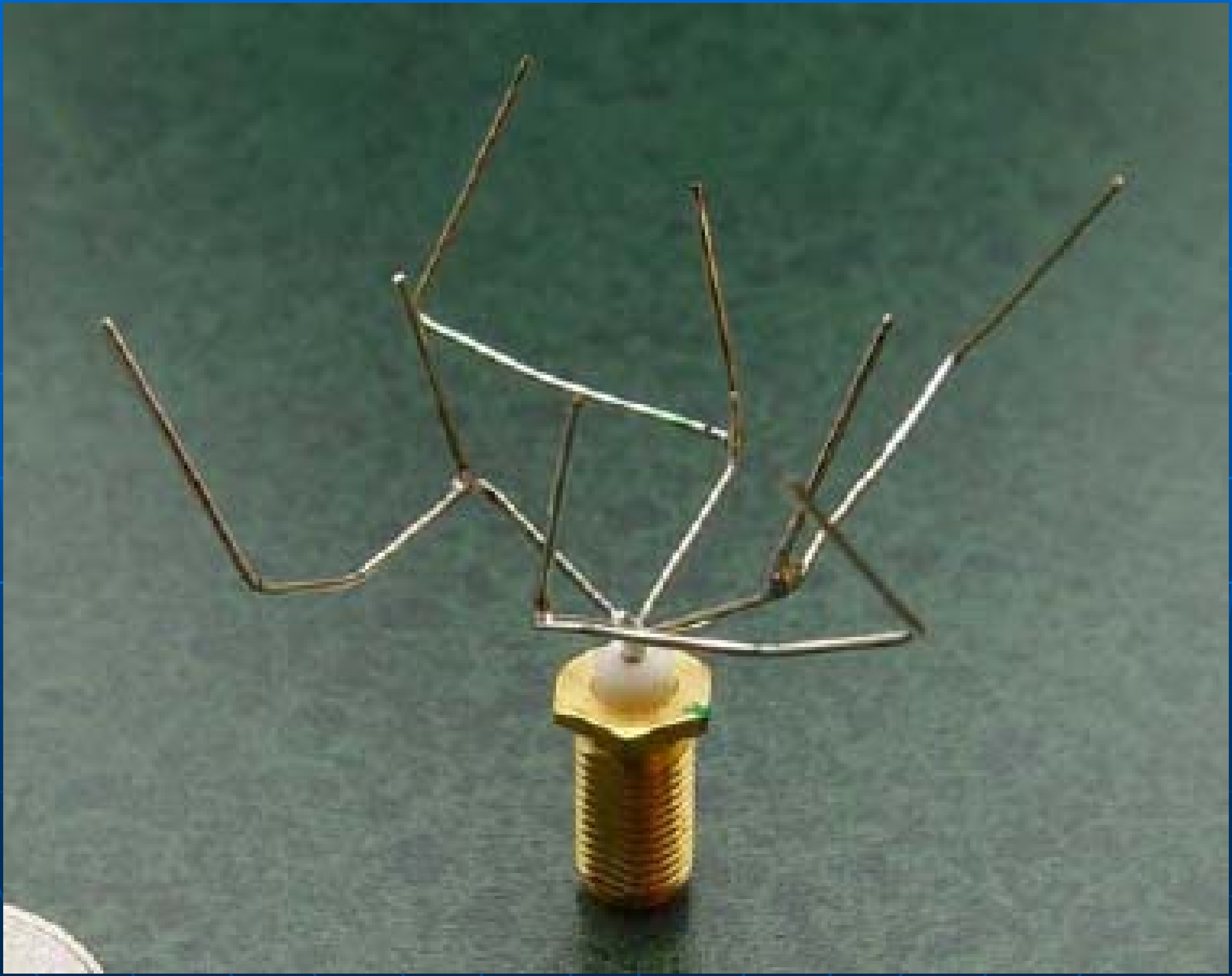
Antennae Again

- Genotype specifies design of 1 arm in 3-space
- Genotype is tree-structured computer program that builds a wire form
- Commands:
  - forward(length radius)
  - rotate_x(angle)
  - rotate_y(angle)
  - rotate_z(angle)
- Branching in genotype → branching in wire form



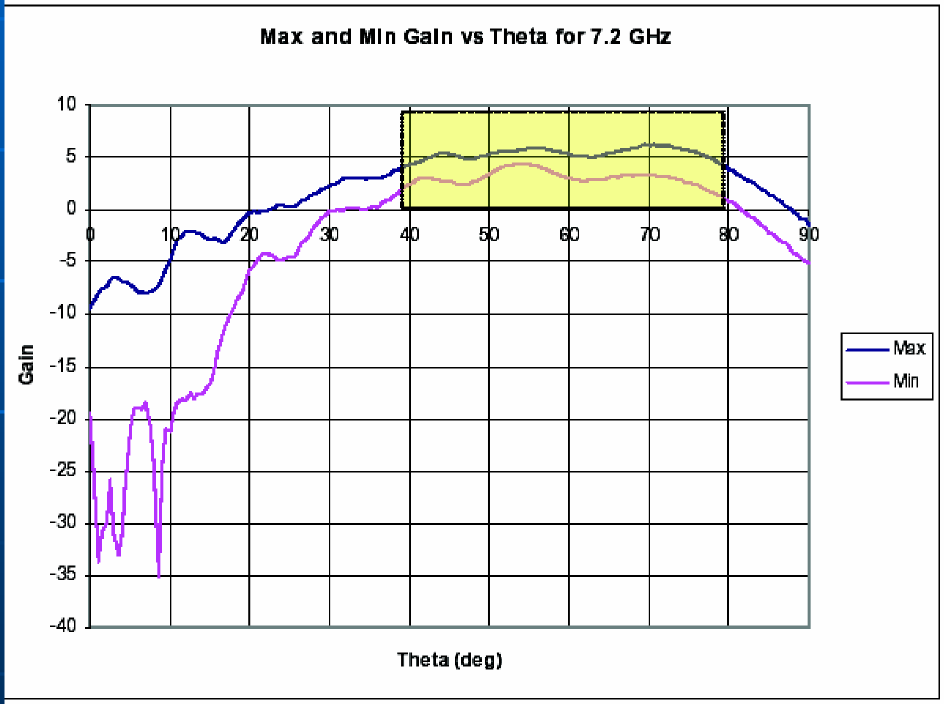


An example antenna

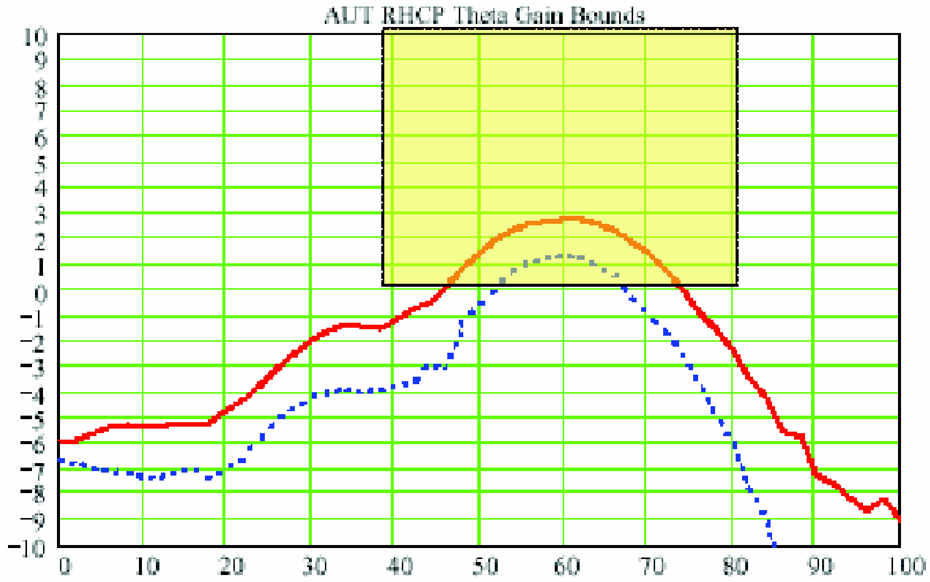


Its performance

Evolved Antenna



Conventional Antenna



Shaded Yellow Box Denotes Area In-Spec, According to Original Mission Requirements