

# **INTRODUCTION TO GENETIC PROGRAMMING**

## **TUTORIAL**

**GECCO-2003  
CHICAGO  
SUNDAY JULY 13, 2003**

**John R. Koza  
Consulting Professor (Medical Informatics)  
Department of Medicine  
School of Medicine**

**Consulting Professor  
Department of Electrical Engineering  
School of Engineering**

**Stanford University  
Stanford, California 94305  
E-MAIL: [koza@stanford.edu](mailto:koza@stanford.edu)  
<http://www.smi.stanford.edu/people/koza/>  
<http://www.genetic-programming.org>  
<http://www.genetic-programming.com>**

## **THE CHALLENGE**

**"How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?"**

**— Attributed to Arthur Samuel (1959)**

## **CRITERION FOR SUCCESS**

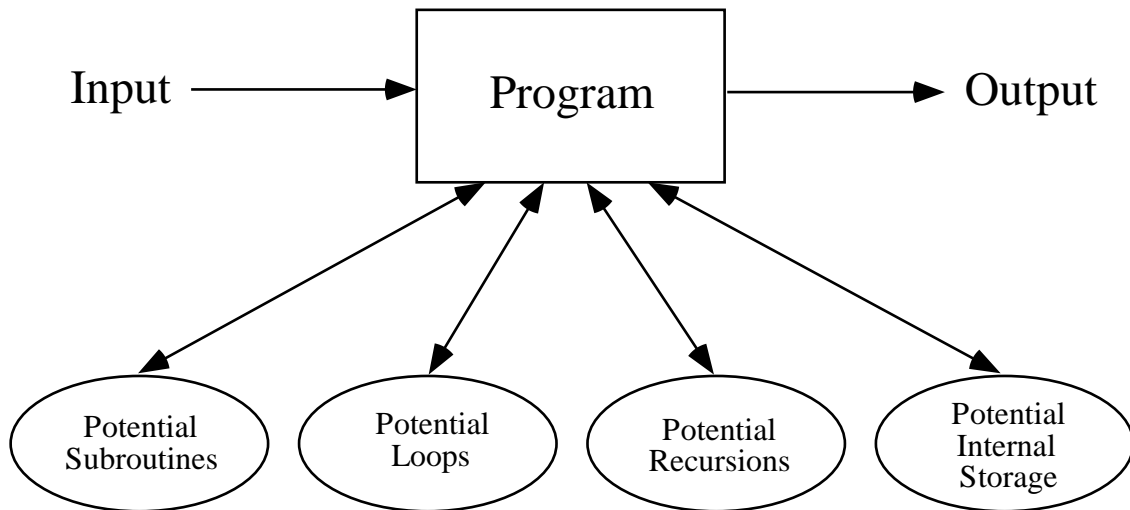
**"The aim [is] ... to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence."**

**— Arthur Samuel (1983)**

# **SOME (OF THE MANY) REPRESENTATIONS USED TO TRY TO ACHIEVE MACHINE INTELLIGENCE IN THE FIELDS OF ARTIFICIAL INTELLIGENCE (AI) AND MACHINE LEARNING (ML)**

- **Decision trees**
- **If-then production rules (e.g., expert system)**
- **Horn clauses**
- **Neural nets (matrices of numerical weights)**
- **Bayesian networks**
- **Frames**
- **Propositional logic**
- **Binary decision diagrams**
- **Formal grammars**
- **Vectors of numerical coefficients for polynomials (adaptive systems)**
- **Tables of values (reinforcement learning)**
- **Conceptual clusters**
- **Concept sets**
- **Parallel if-then rules (e.g., genetic classifier systems)**

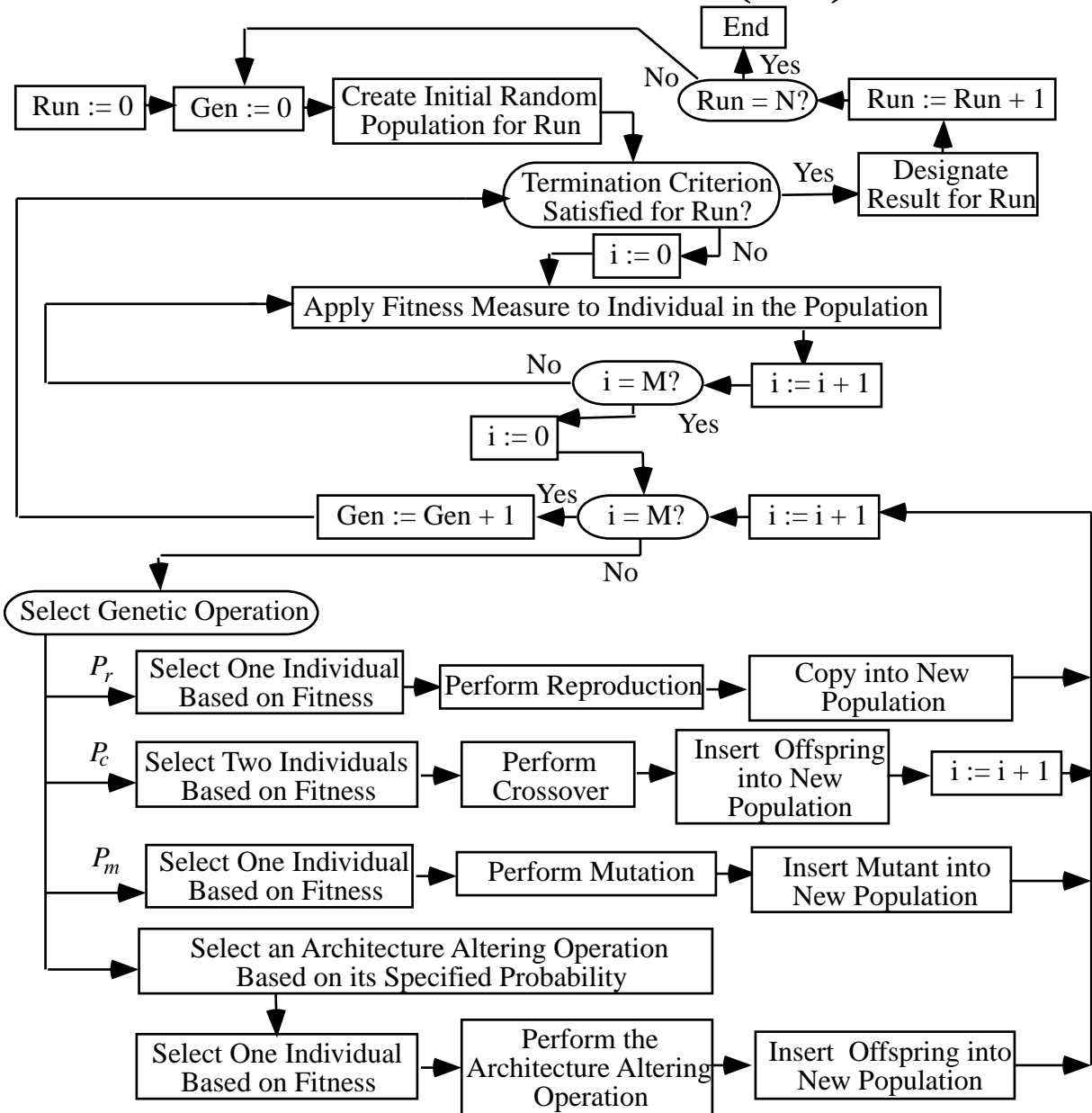
# A COMPUTER PROGRAM



## REPRESENTATION

- "Our view is that computer programs are the best representation of computer programs."

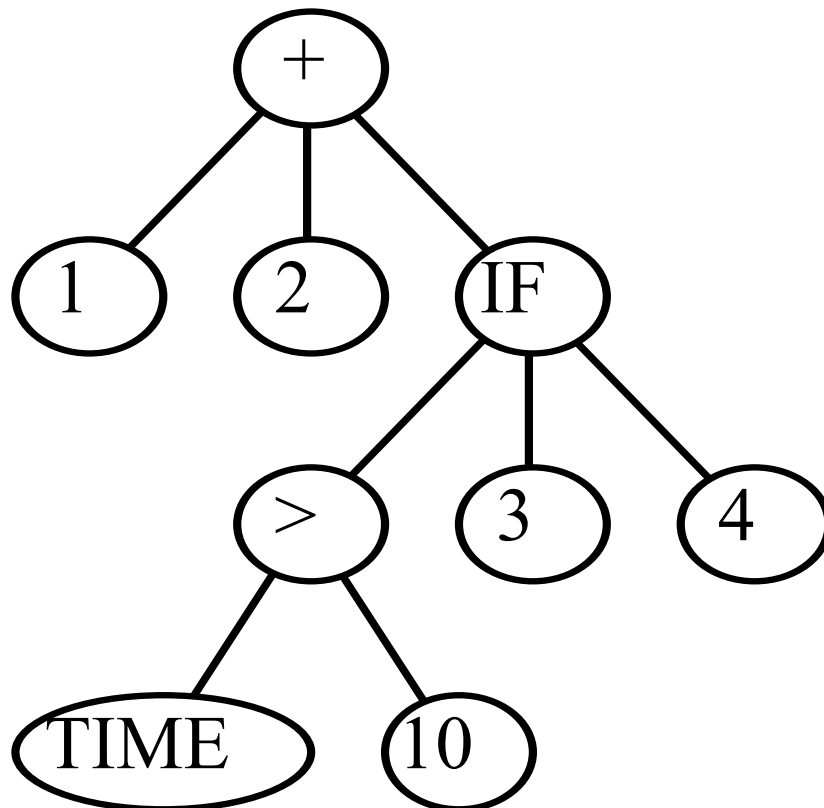
# FLOWCHART FOR GENETIC PROGRAMMING (GP)



**COMPUTER PROGRAM  
= PARSE TREE = PROGRAM TREE  
= PROGRAM IN LISP = DATA = LIST**

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

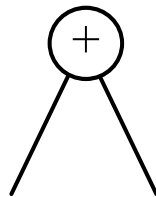
- **ATOMS (TERMINALS) = 1, 2, 10, 3, 4, TIME**
- **FUNCTIONS = +, IF, >**



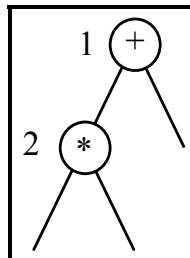
# EXAMPLE OF RANDOM CREATION OF A PROGRAM TREE

- Terminal set  $T = \{A, B, C\}$
- Function set  $F = \{+, -, *, \%, \text{IFLTE}\}$

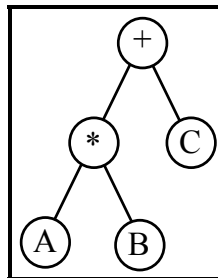
**BEGIN WITH TWO-ARGUMENT +**



**CONTINUE WITH TWO-ARGUMENT \***



**COMPLETE WITH TERMINALS A, B,  
AND C**

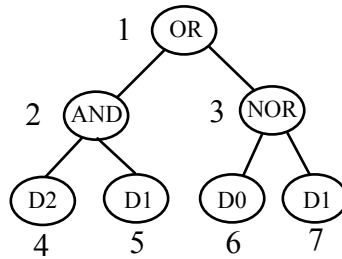


- The result is a syntactically valid executable program (provided the set of functions is closed)

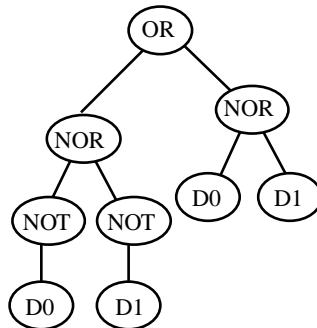
## MUTATION OPERATION

- Select parent probabilistically based on fitness
- Pick point from 1 to NUMBER-OF-POINTS
- Delete subtree at the picked point
- Grow new subtree at the mutation point in same way as generated trees for initial random population (generation 0)
- The result is a syntactically valid executable program

## ONE PARENTAL PROGRAM



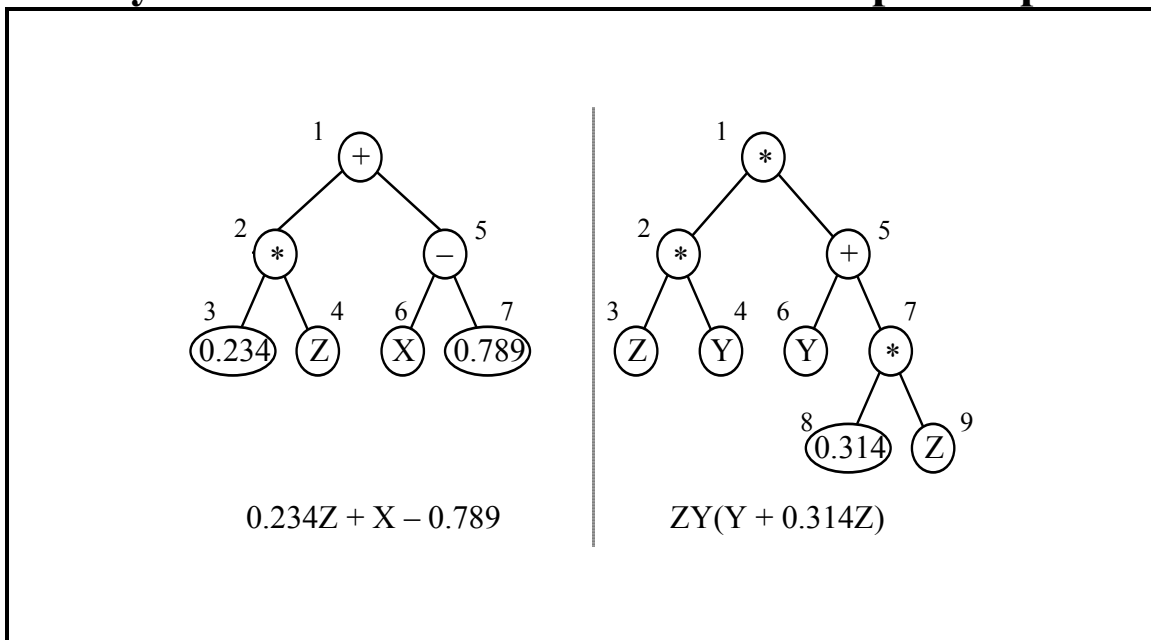
## OFFSPRING PRODUCED BY MUTATION





# CROSSOVER (SEXUAL RECOMBINATION) OPERATION FOR COMPUTER PROGRAMS

- Select two parents probabilistically based on fitness
- Randomly pick a number from 1 to NUMBER-OF-POINTS – independently for each of the two parental programs
- Identify the two subtrees rooted at the two picked points



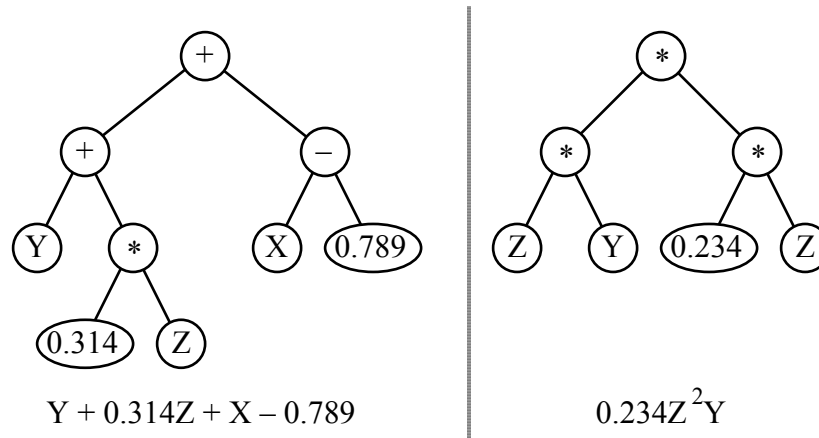
Parent 1:

$(+ \ (\underline{* \ 0.234 \ Z}) \ (- \ X \ 0.789))$

Parent 2:

$(* \ (* \ Z \ Y) \ (\underline{+ \ Y \ (* \ 0.314 \ Z)}))$

## THE CROSSOVER OPERATION (TWO OFFSPRING VERSION)



**Offspring 1:**

$$\left( + \left( + Y \left( * 0.314 Z \right) \right) \right. \\ \left. \left( - X 0.789 \right) \right)$$

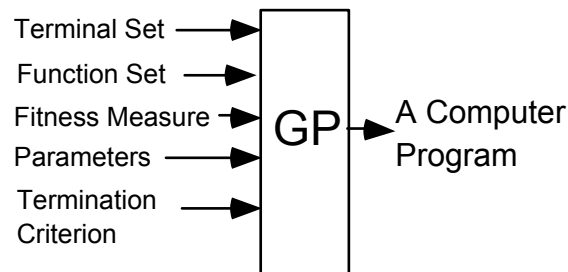
**Offspring 2:**

$$\left( * \left( * Z Y \right) \left( * 0.234 Z \right) \right)$$

- The result is a syntactically valid executable program

## 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**
  - population size
  - number of generations
  - minor parameters
- **Determining the method for designating a result and the criterion for terminating a run**

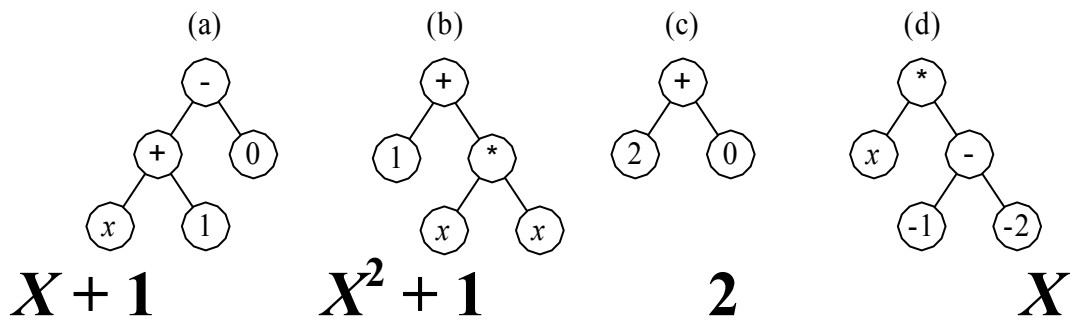


**(ABBREVIATED) TABLEAU FOR  
SYMBOLIC REGRESSION OF  
QUADRATIC POLYNOMIAL  $x^2 + x + 1$**

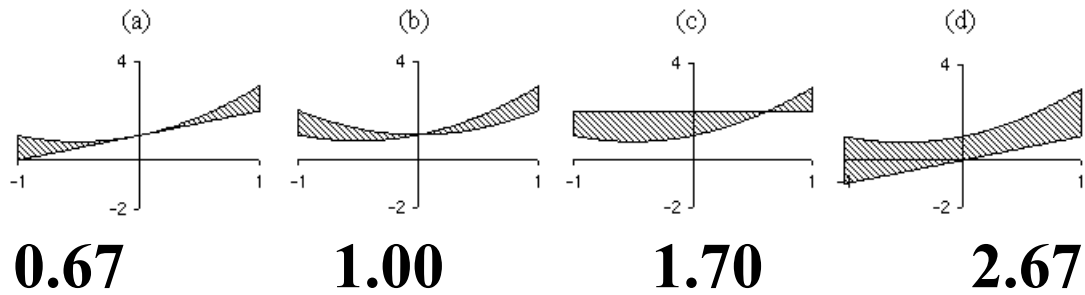
	<b>Objective:</b>	Find a computer program with one input (independent variable $x$ ), whose output equals the value of the quadratic polynomial $x^2 + x + 1$ in range from -1 to +1.
<b>1</b>	<b>Terminal set:</b>	$\mathbf{T} = \{x\}$
<b>2</b>	<b>Function set:</b>	$\mathbf{F} = \{+, -, *, \%$ NOTE: The protected division function $\%$ returns a value of 1 when division by 0 is attempted (including 0 divided by 0)
<b>3</b>	<b>Fitness:</b>	The sum of the absolute value of the differences (errors), computed (in some way) over values of the independent variable $x$ from -1.0 to +1.0, between the program's output and the target quadratic polynomial $x^2 + x + 1$ .
<b>4</b>	<b>Parameters:</b>	Population size $M = 4$ .
<b>5</b>	<b>Termination:</b>	An individual emerges whose sum of absolute errors is less than 0.1

# SYMBOLIC REGRESSION OF QUADRATIC POLYNOMIAL $X^2 + X + 1$

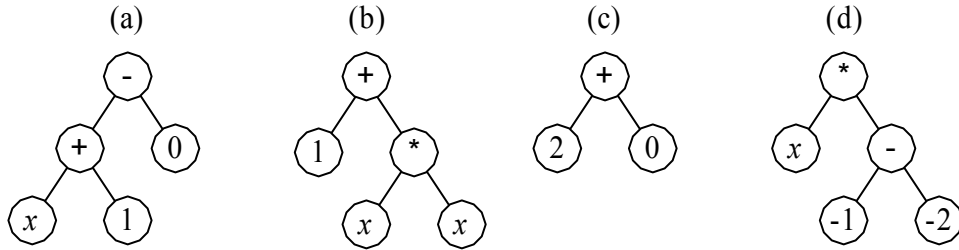
## INITIAL POPULATION OF FOUR RANDOMLY CREATED INDIVIDUALS OF GENERATION 0



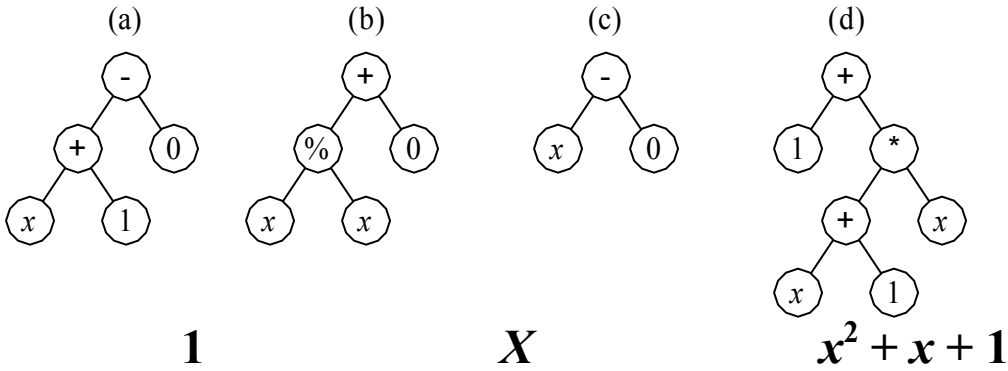
### FITNESS



# SYMBOLIC REGRESSION OF QUADRATIC POLYNOMIAL $X^2 + X + 1$



## GENERATION 1



**Copy of (a)**

**Mutant of (c)**  
 —picking “2”  
 as mutation  
 point

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

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

**SYMBOLIC REGRESSION  
OF QUARTIC POLYNOMIAL  $X^4+X^3+X^2+X$   
(WITH 21 FITNESS CASES)**

Independent variable (Input)	$X$	Dependent Variable (Output)	$Y$
-1.0		0.0000	
-0.9		-0.1629	
-0.8		-0.2624	
-0.7		-0.3129	
-0.6		-0.3264	
-0.5		-0.3125	
-0.4		-0.2784	
-0.3		-0.2289	
-0.2		-0.1664	
-0.1		-0.0909	
0		0.0	
0.1		0.1111	
0.2		0.2496	
0.3		0.4251	
0.4		0.6496	
0.5		0.9375	
0.6		1.3056	
0.7		1.7731	
0.8		2.3616	
0.9		3.0951	
1.0		4.0000	

## TABLEAU — SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

<b>Objective:</b>	Find a function of one independent variable, in symbolic form, that fits a given sample of 21 $(x_i, y_i)$ data points
<b>Terminal set:</b>	$x$ (the independent variable).
<b>Function set:</b>	$+$ , $-$ , $*$ , $\%$ , $\text{SIN}$ , $\text{COS}$ , $\text{EXP}$ , $\text{RLOG}$
<b>Fitness cases:</b>	The given sample of 21 data points $(x_i, y_i)$ where the $x_i$ are in interval $[-1,+1]$ .
<b>Raw fitness:</b>	The sum, taken over the 21 fitness cases, of the absolute value of difference between value of the dependent variable produced by the individual program and the target value $y_i$ of the dependent variable.
<b>Standardized fitness:</b>	Equals raw fitness.
<b>Hits:</b>	Number of fitness cases (0 – 21) for which the value of the dependent variable produced by the individual program comes within 0.01 of the target value $y_i$ of the dependent variable.
<b>Wrapper:</b>	None.
<b>Parameters:</b>	Population size, $M = 500$ . Maximum number of generations to be run, $G = 51$ .
<b>Success Predicate:</b>	An individual program scores 21 hits.



**SYMBOLIC REGRESSION  
OF QUARTIC POLYNOMIAL  $X^4+X^3+X^2+X$**

**WORST-OF-GENERATION INDIVIDUAL  
IN GENERATION 0 WITH RAW FITNESS  
OF  $10^{38}$**

**(EXP (- (% X (- X (SIN X))))  
(RLOG (RLOG (\* X X))))))**

**Equivalent to**

$$e^x / (x - \sin x) - \log \log x^*x$$

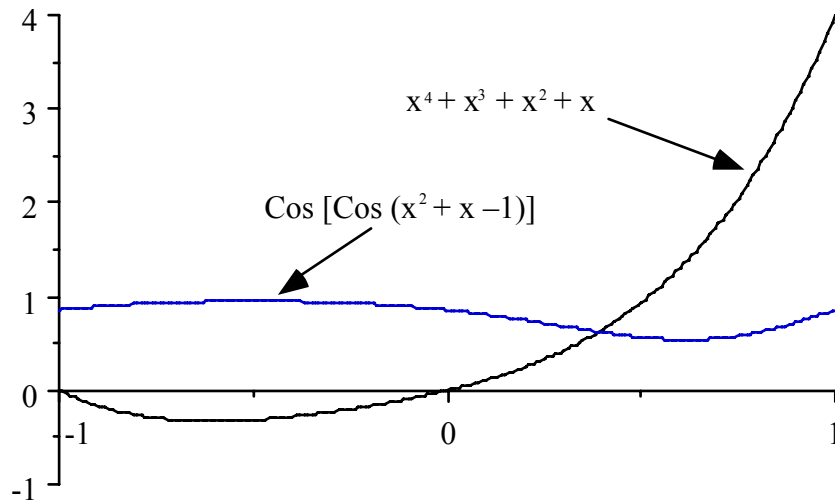
# SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

**MEDIAN INDIVIDUAL IN GENERATION  
0 WITH RAW FITNESS OF 23.67  
(AVERAGE ERROR OF 1.3)**

**(COS (COS (+ (- (\* X X) (% X  
X)) X)))**

Equivalent to

$$\text{Cos} [\text{Cos} (x^2 + x - 1)]$$



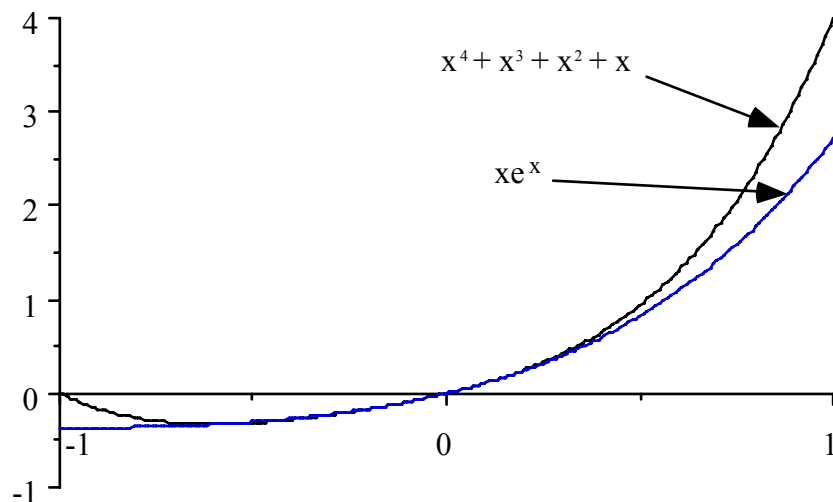
# SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

**BEST-OF-GENERATION INDIVIDUAL IN  
GENERATION 0 WITH RAW FITNESS OF  
4.47 (AVERAGE ERROR OF 0.2)**

```
( * X ( + ( + ( - ( % X X ) ( % X X ) )
(SIN ( - X X ) ) ) ( RLOG ( EXP ( EXP
X ) ) ) ) ) )
```

Equivalent to

$$xe^x$$



# SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

## CREATION OF GENERATION 1 FROM GENERATION 0

- In the so-called "generational" model for genetic algorithms, a new population is created that is equal in size to the old population
  - 1% mutation (i.e., 5 individuals out of 500)
  - 9% reproduction (i.e., 45 individuals)
  - 90% crossover (i.e., 225 pairs of parents — yielding 450 offspring)
- All participants in mutation, reproduction, and crossover are chosen from the current population **PROBABILISTICALLY, BASED ON FITNESS**
  - Anything can happen
  - Nothing is guaranteed
  - The search is heavily (but not completely) biased toward high-fitness individuals
  - The best is not guaranteed to be chosen
  - The worst is not necessarily excluded
  - Some (but not much) attention is given even to low-fitness individuals

**SYMBOLIC REGRESSION  
OF QUARTIC POLYNOMIAL  $X^4+X^3+X^2+X$**

**BEST-OF-GENERATION INDIVIDUAL IN  
GENERATION 2 WITH RAW FITNESS OF  
2.57 (AVERAGE ERROR OF 0.1)**

**(+ (\* (\* (+ X (\* X (\* X (% (% X  
X) (+ X X))))))  
(+ X (\* X X))) X) X)**

**Equivalent to...**

$$x^4 + 1.5x^3 + 0.5x^2 + x$$

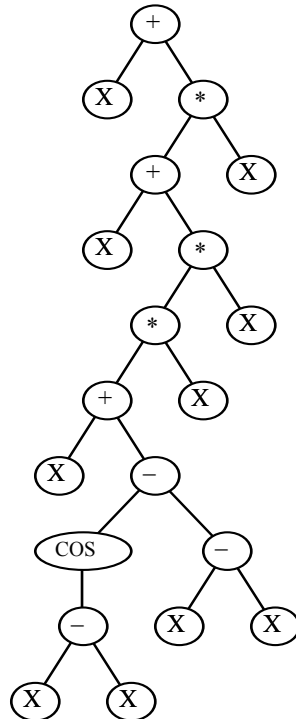
# SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4 + X^3 + X^2 + X$

**BEST-OF-RUN INDIVIDUAL IN  
GENERATION 34 WITH RAW FITNESS  
OF 0.00 (100%-CORRECT)**

**( + X ( \* ( + X ( \* ( \* ( + X ( - ( COS  
( - X X ) ) ( - X X ) ) ) X ) X ) ) X ) )**

Equivalent to

$$x^4 + x^3 + x^2 + x$$

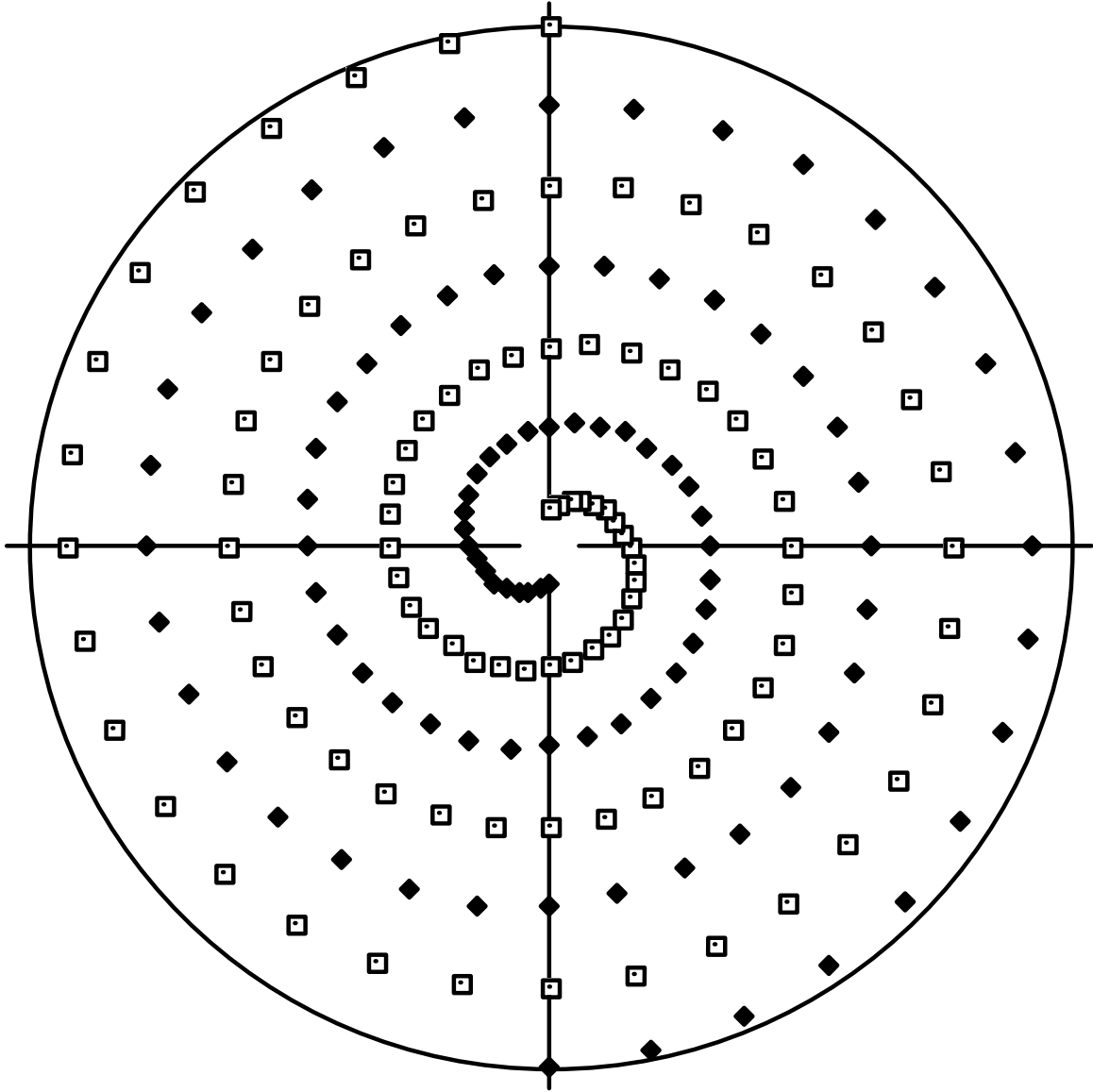


# SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

## OBSERVATIONS

- GP works on this problem
- GP determines the size and shape of the solution
  - number of operations needed to solve the problem
  - size and shape of the program tree
  - content of the program tree (i.e., sequence of operations)
- GP operates the same whether the solution is linear, polynomial, a rational fraction of polynomials, exponential, trigonometric, etc.
- It's not how a human programmer would have done it
  - $\text{Cos}(X - X) = 1$
  - Not parsimonious
- The extraneous functions – SIN, EXP, RLOG, and RCOS are absent in the best individual of later generations because they are detrimental
  - $\text{Cos}(X - X) = 1$  is the exception that proves the rule
- The answer is algebraically correct (hence no further cross validation is needed)

# CLASSIFICATION PROBLEM INTER-TWINED SPIRALS



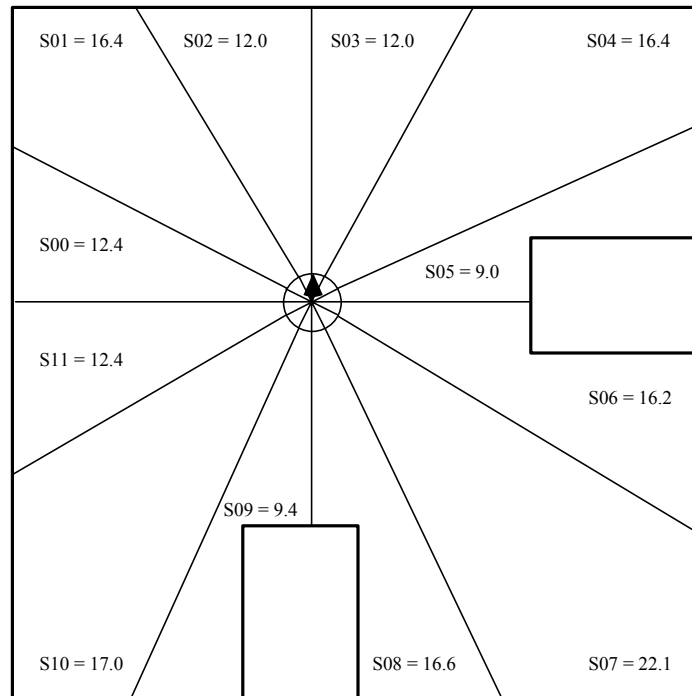


## GP TABLEAU – INTERTWINED SPIRALS

<b>Objective:</b>	Find a program to classify a given point in the $x$ - $y$ plane to the red or blue spiral.
<b>Terminal set:</b>	$x$ , $y$ , $\mathfrak{R}$ , where $\mathfrak{R}$ is the ephemeral random floating-point constant ranging between $-1.000$ and $+1.000$ .
<b>Function set:</b>	$+$ , $-$ , $*$ , $\%$ , IFLTE, SIN, COS.
<b>Fitness cases:</b>	194 points in the $x$ - $y$ plane.
<b>Raw fitness:</b>	The number of correctly classified points (0 – 194)
<b>Standardized fitness:</b>	The maximum raw fitness (i.e., 194) minus the raw fitness.
<b>Hits:</b>	Equals raw fitness.
<b>Wrapper:</b>	Maps any individual program returning a positive value to class +1 (red) and maps all other values to class $-1$ (blue).
<b>Parameters:</b>	$M = 10,000$ (with over-selection). $G = 51$ .
<b>Success predicate:</b>	An individual program scores 194 hits.

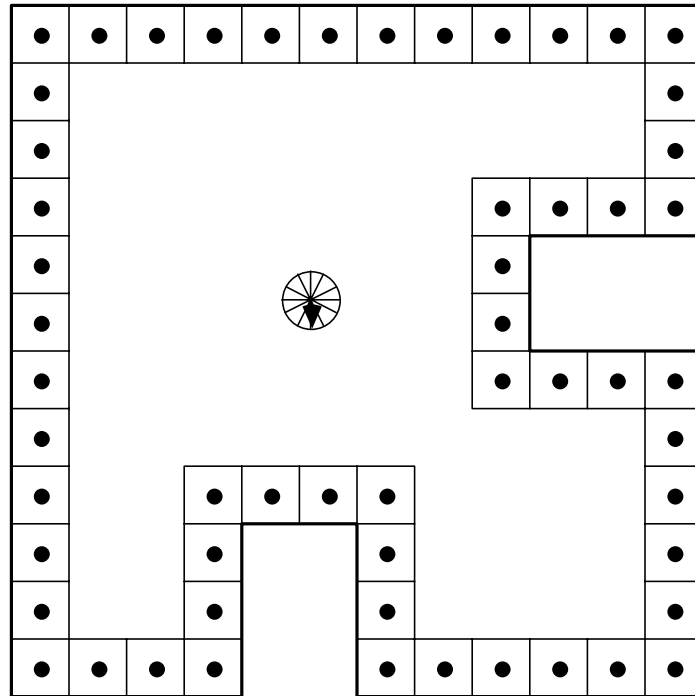
# WALL-FOLLOWING PROBLEM

## 12 SONAR SENSORS



# WALL-FOLLOWING PROBLEM

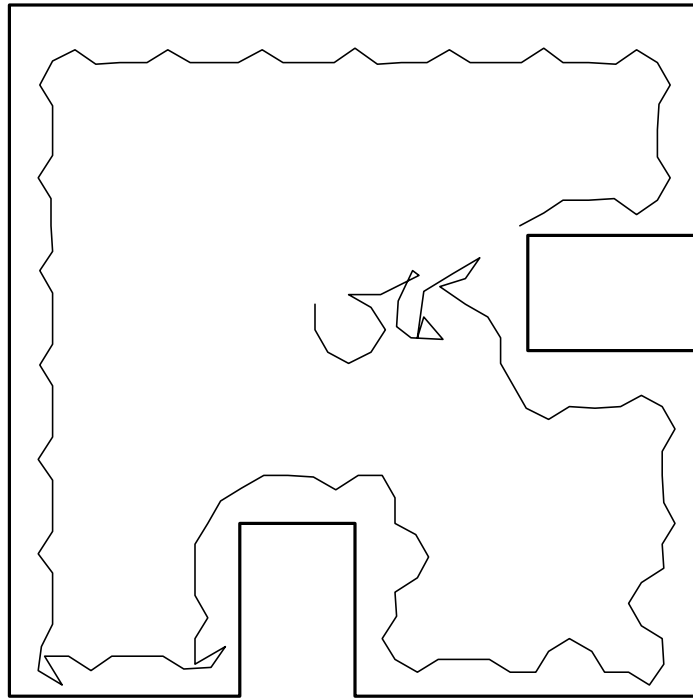
## FITNESS MEASURE



# WALL-FOLLOWING PROBLEM

## BEST PROGRAM OF GENERATION 57

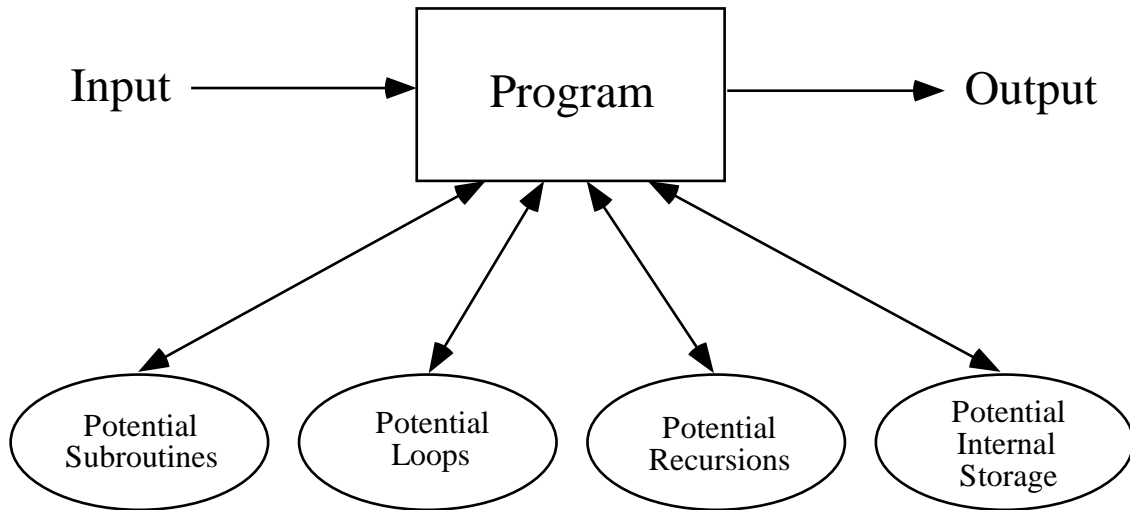
- Scores 56 hits (out of 56)
- 145point program tree



**24 PROBLEMS SHOWN IN 1992  
VIDEOTAPE  
*GENETIC PROGRAMMING: THE MOVIE*  
(KOZA AND RICE 1992)**

- **Symbolic Regression**
- **Intertwined Spirals**
- **Artificial Ant**
- **Truck Backer Upper**
- **Broom Balancing**
- **Wall Following**
- **Box Moving**
- **Discrete Pursuer-Evader Game**
- **Differential Pursuer-Evader Game**
- **Co-Evolution of Game-Playing Strategies**
- **Inverse Kinematics**
- **Emergent Collecting**
- **Central Place Foraging**
- **Block Stacking**
- **Randomizer**
- **1-D Cellular Automata**
- **2-D Cellular Automata**
- **Task Prioritization**
- **Programmatic Image Compression**
- **Finding  $3\sqrt{2}$**
- **Econometric Exchange Equation**
- **Optimization (Lizard)**
- **Boolean 11-Multiplexer**
- **11-Parity–Automatically Defined Functions**

# AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)



- **Subroutines provide one way to REUSE code — possibly with different instantiations of the dummy variables (formal parameters)**
- **Loops (and iterations) provide a 2<sup>nd</sup> way to REUSE code**
- **Recursion provide a 3<sup>rd</sup> way to REUSE code**
- **Memory provides a 4<sup>th</sup> way — to REUSE the results of executing code**

## AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

**10 FITNESS-CASES SHOWING THE  
VALUE OF THE DEPENDENT  
VARIABLE,  $D$ , ASSOCIATED WITH THE  
VALUES OF THE SIX INDEPENDENT  
VARIABLES,  $L_0, W_0, H_0, L_1, W_1, H_1$**

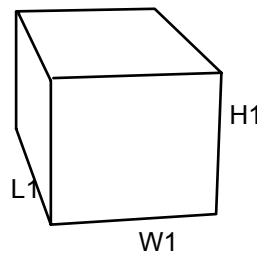
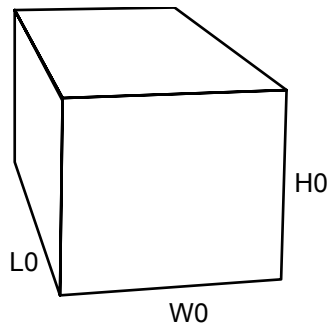
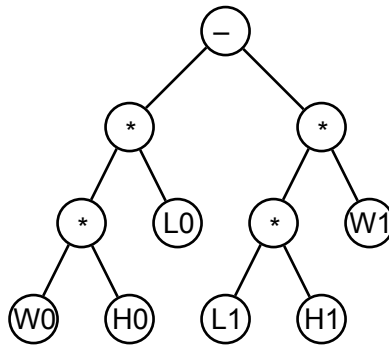
<b>Fitness case</b>	$L_0$	$W_0$	$H_0$	$L_1$	$W_1$	$H_1$	<b>Dependent variable <math>D</math></b>
<b>1</b>	3	4	7	2	5	3	<b>54</b>
<b>2</b>	7	10	9	10	3	1	<b>600</b>
<b>3</b>	10	9	4	8	1	6	<b>312</b>
<b>4</b>	3	9	5	1	6	4	<b>111</b>
<b>5</b>	4	3	2	7	6	1	<b>-18</b>
<b>6</b>	3	3	1	9	5	4	<b>-171</b>
<b>7</b>	5	9	9	1	7	6	<b>363</b>
<b>8</b>	1	2	9	3	9	2	<b>-36</b>
<b>9</b>	2	6	8	2	6	10	<b>-24</b>
<b>10</b>	8	1	10	7	5	1	<b>45</b>

# AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

## SOLUTION WITHOUT ADFs

```
( - ( * ( * W0 L0 ) H0 )
    ( * ( * W1 L1 ) H1 ) )
```

$$D = W0 * L0 * H0 - W1 * L1 * H1$$

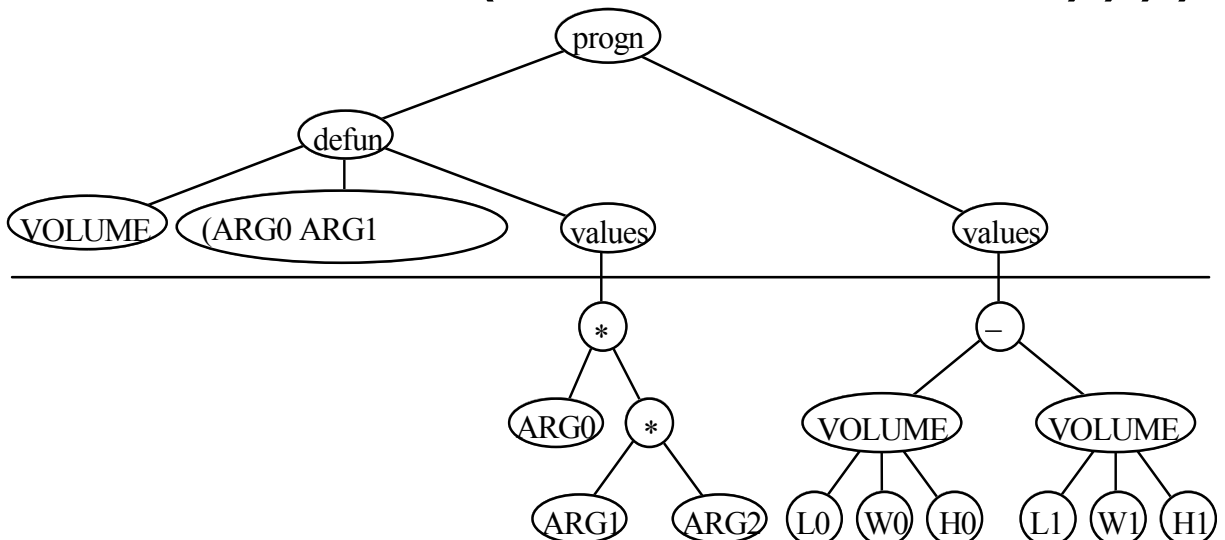




# AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

AN OVERALL COMPUTER PROGRAM  
CONSISTING OF ONE FUNCTION-  
DEFINING BRANCH (ADF,  
SUBROUTINE) AND ONE RESULT-  
PRODUCING BRANCH (MAIN  
PROGRAM)

```
(progn
  (defun volume (arg0 arg1 arg2)
    (values
      (* arg0 (* arg1 arg2))))
  (values (- (volume L0 W0 H0)
            (volume L1 W1 H1))))
```



## AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

**IF WE ADD TWO NEW VARIABLES FOR  
VOLUME ( $V_0$  AND  $V_1$ ), THE 6-  
DIMENSIONAL NON-LINEAR  
REGRESSION PROBLEM BECOMES AN  
8-DIMENSIONAL PROBLEM**

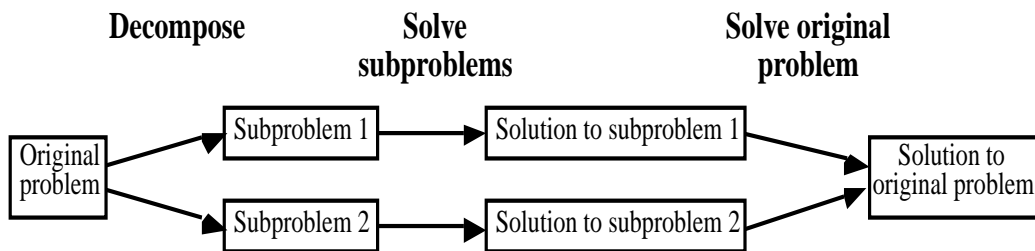
<b>Fitness case</b>	$L_0$	$W_0$	$H_0$	$L_1$	$W_1$	$H_1$	$V_0$	$V_1$	$D$
<b>1</b>	3	4	7	2	5	3	<b>84</b>	<b>30</b>	<b>54</b>
<b>2</b>	7	10	9	10	3	1	<b>630</b>	<b>30</b>	<b>600</b>
<b>3</b>	10	9	4	8	1	6	<b>360</b>	<b>48</b>	<b>312</b>
<b>4</b>	3	9	5	1	6	4	<b>135</b>	<b>24</b>	<b>111</b>
<b>5</b>	4	3	2	7	6	1	<b>24</b>	<b>42</b>	<b>-18</b>
<b>6</b>	3	3	1	9	5	4	<b>9</b>	<b>180</b>	<b>-171</b>
<b>7</b>	5	9	9	1	7	6	<b>405</b>	<b>42</b>	<b>363</b>
<b>8</b>	1	2	9	3	9	2	<b>18</b>	<b>54</b>	<b>-36</b>
<b>9</b>	2	6	8	2	6	10	<b>96</b>	<b>120</b>	<b>-24</b>
<b>10</b>	8	1	10	7	5	1	<b>80</b>	<b>35</b>	<b>45</b>

- However, the problem can now be approached as a 2-dimensional LINEAR regression problem.

# AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

## TOP-DOWN VIEW OF THREE STEP HIERARCHICAL PROBLEM-SOLVING PROCESS

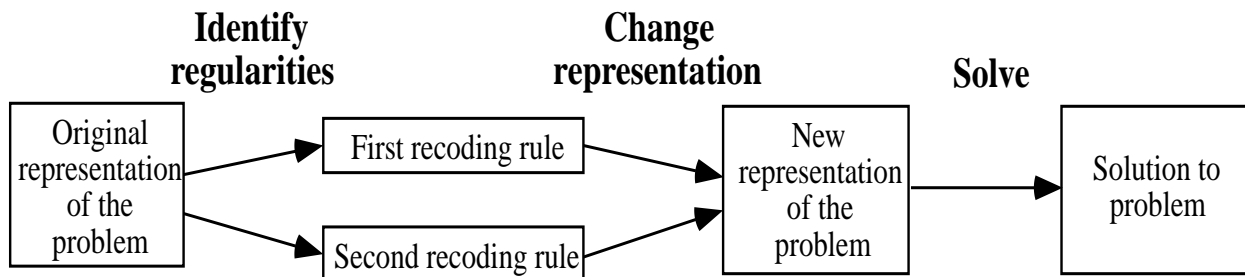
### DIVIDE AND CONQUER



- **Decompose a problem into subproblems**
- **Solve the subproblems**
- **Assemble the solutions of the subproblems into a solution for the overall problem**

# AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

## BOTTOM-UP VIEW OF THREE STEP HIERARCHICAL PROBLEM-SOLVING PROCESS



- **Identify regularities**
- **Change the representation**
- **Solve the overall problem**

## **AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)**

- In generation 0, we create a population of programs, each consisting of a main result-producing branch (RPB) and one or more function-defining branches (automatically defined functions, ADFs, subroutines)
  - Different ingredients for RPB and ADFs
  - The terminal set of an ADF typically contains dummy arguments (formal parameters), such as `ARG0`, `ARG1`, ...
  - The function set of the RPB contains `ADF0`, ...
  - ADFs are private and associated with a particular individual program in the population
- The entire program is executed and evaluated for fitness
- Genetic operation of reproduction is the same as before
- Mutation operation starts (as before) by picking a mutation point from either RPB or an ADF and deleting the subtree rooted at that point. As before, a subtree is then grown at the point. The new subtree is composed of the allowable ingredients for that point — so that the result is a syntactically valid executable program.
- Crossover operation starts (as before) by picking a crossover point from either RPB or an ADF of one parent. The choice of crossover point in the second parent is then restricted (e.g., to the RPB or to the ADF) — so that when the subtrees are swapped, the result is a syntactically valid executable program.

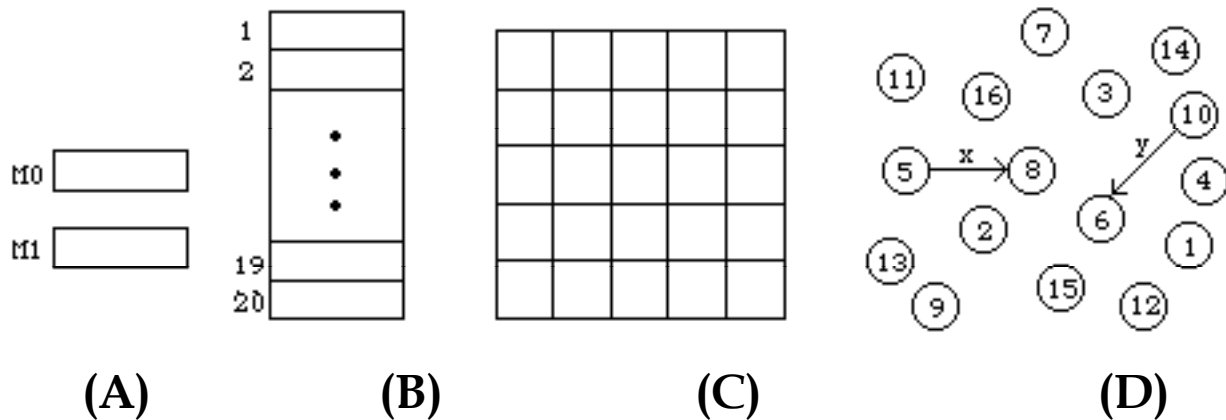
# **AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)**

## **8 MAIN POINTS FROM BOOK *GENETIC PROGRAMMING II: AUTOMATIC DISCOVERY OF REUSABLE PROGRAMS (KOZA 1994)***

- **ADFs work.**
- **ADFs do not solve problems in the style of human programmers.**
- **ADFs reduce the computational effort required to solve a problem.**
- **ADFs usually improve the parsimony of the solutions to a problem.**
- **As the size of a problem is scaled up, the size of solutions increases more slowly with ADFs than without them.**
- **As the size of a problem is scaled up, the computational effort required to solve a problem increases more slowly with ADFs than without them.**
- **The advantages in terms of computational effort and parsimony conferred by ADFs increase as the size of the problem is scaled up.**

# REUSE

## MEMORY AND STORAGE



- (A) Settable (named) variables (*Genetic Programming*, Koza 1992) using setting (writing) functions (`SETM0 X`) and (`SETM1 Y`) and reading by means of terminals `M0` and `M1`.
- (B) Indexed memory similar to linear (vector) computer memory (Teller 1994) using (`READ K`) and (`WRITE X K`)
- (C) Matrix memory (Andre 1994)
- (D) Relational memory (Brave 1995, 1996)

## LANGDON'S DATA STRUCTURES

- Stacks
- Queues
- Lists
- Rings

# REUSE

## AUTOMATICALLY DEFINED ITERATIONS (ADIs)

- Overall program consisting of an automatically defined function **ADF0**, an iteration-performing branch **IPB0**, and a result-producing branch **RPB0**.
- Iteration is over a known, fixed set
  - protein or DNA sequence (of varying length)
  - time-series data
  - two-dimensional array of pixels



# REUSE

## TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM

- Goal is to classify a given protein segment as being a transmembrane domain or non-transmembrane area of the protein
  - Generation 20 — Run 3 — Subset-creating version
    - in-sample correlation of 0.976
    - out-of-sample correlation of 0.968
    - out-of-sample error rate 1.6%
- (progn

```
(defun ADF0 ()
  (ORN (ORN (ORN (I?) (H?)) (ORN (P?) (G?))) (ORN (ORN
    (ORN (Y?) (N?)) (ORN (T?) (Q?))) (ORN (A?) (H?))))))
```

```
(defun ADF1 ()
  (values (ORN (ORN (ORN (A?) (I?)) (ORN (L?) (W?)))
    (ORN (ORN (T?) (L?)) (ORN (T?) (W?))))))
```

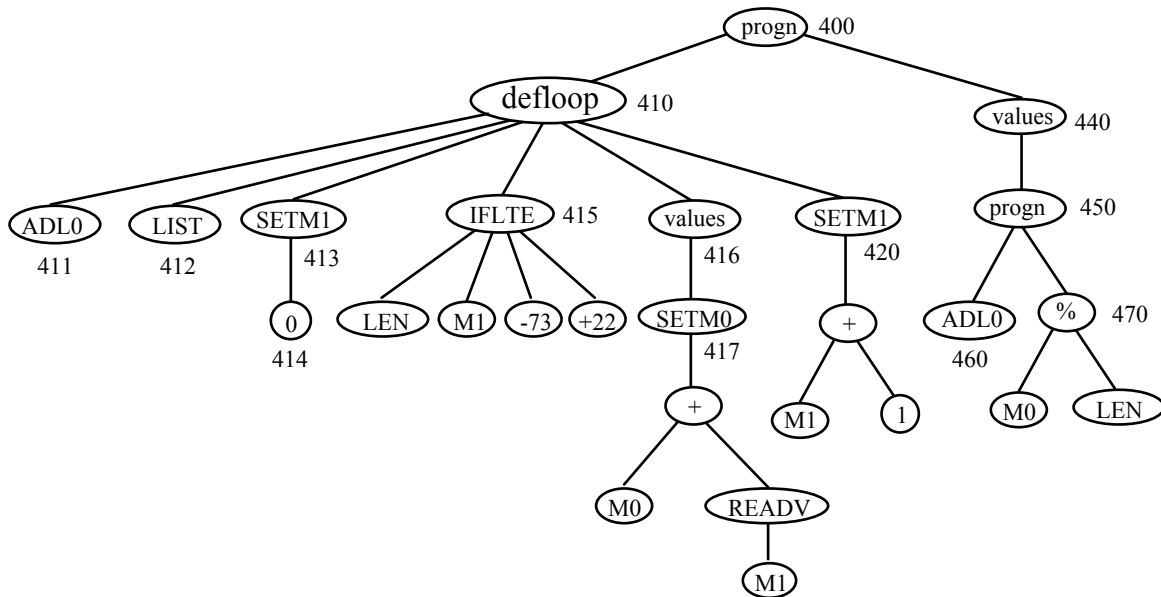
```
(defun ADF2 ()
  (values (ORN (ORN (ORN (ORN (ORN (D?) (E?)) (ORN (ORN
    (ORN (D?) (E?)) (ORN (ORN (T?) (W?)) (ORN (Q?)
    (D?)))) (ORN (K?) (P?)))) (ORN (K?) (P?)) (ORN (T?)
    (W?))) (ORN (ORN (E?) (A?)) (ORN (N?) (R?))))))
```

```
(progn (loop-over-residues
  (SETM0 (+ (- (ADF1) (ADF2)) (SETM3 M0))))
```

```
(values (% (% M3 M0) (% (% (% (- L -0.53) (* M0
  M0)) (+ (% (% M3 M0) (% (+ M0 M3) (% M1 M2))) M2)) (%
  M3 M0))))))
```

# REUSE

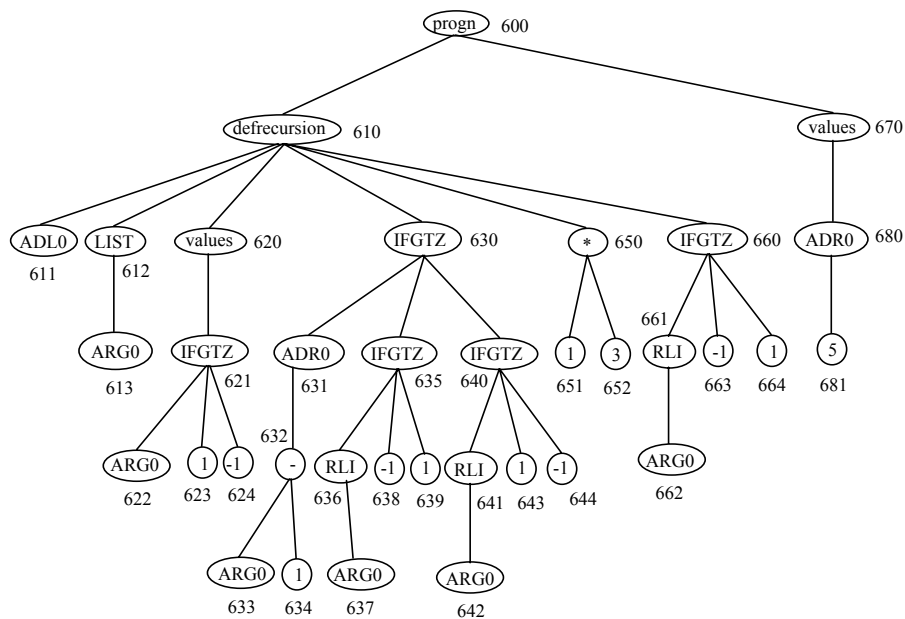
## EXAMPLE OF A PROGRAM WITH A FOUR-BRANCH AUTOMATICALLY DEFINED LOOP (ADL0) AND A RESULT- PRODUCING BRANCH



# REUSE

## AUTOMATICALLY DEFINED RECURSION (ADRO) AND A RESULT- PRODUCING BRANCH

- a recursion condition branch, RCB
- a recursion body branch, RBB
- a recursion update branch, RUB
- a recursion ground branch, RGB



## GP TECHNIQUES

- control structures involving multiple result-producing branches (Luke and Spector 1996a Bennett 1996a Svingen 1997)
- adaptive self-modifying ontogenetic genetic programming (Spector and Stoffel 1996a 1996b)
- cultural storage and transmission (Spector and Luke 1996a 1996b)
- hierarchical problem solving (Rosca and Ballard 1994a 1994b; Rosca 1995; Rosca 1997)
- modules (Angeline and Pollack 1993 1994; Angeline 1993 1994; Kinnear 1994b)
- logic grammars (Wong and Leung 1995a 1995b 1995c 1995d 1995e 1995f 1997)
- cellular encoding (developmental genetic programming) for evolving neural networks (Gruau 1992a 1992b 1993 1994a 1994b; Gruau and Whitley 1993; Esparcia-Alcazar and Sharman 1997)
- developmental methods for evolving finite automata using genetic programming (Brave 1996a)
- developmental methods for shape optimization (Kennelly 1997)
- evolving graphs and networks (Luke and Spector 1996b)
- using a grammar to represent bias and background knowledge (Whigham 1995a 1995b 1996)
- developmental methods for fuzzy logic systems (Tunstel and Jamshidi 1996)

## GP TECHNIQUES — CONTINUED

- **diploidy and dominance (Greene 1997a 1997b)**
- **Turing completeness of genetic programming (Teller 1994c; Nordin and Banzhaf 1995)**
- **evolution of chemical topological structures (Nachbar 1997 1998)**
- **interactive fitness measures (Poli and Cagnoni 1997;) and in particular in graphics and art (Sims 1991a 1991b 1992a 1992b 1993)**
- **variations in crossover operations (Poli and Langdon 1997)**
- **distributed processes and multi-agent systems (Haynes Sen Schoenefeld and Wainwright 1995; Ryan 1995; Luke and Spector 1996a; Iba 1996; Iba Nozoe and Ueda 1997; Qureshi 1996; Crosbie and Spafford 1995)**
- **complexity-based fitness measures using minimum description length (Iba Kurita de Garis and Sato 1993; Iba deGaris and Sato 1994)**
- **co-evolution (Reynolds 1994c)**
- **steady state genetic programming (Reynolds 1993 1994a 1994b)**
- **use of noise in fitness cases (Reynolds 1994d)**
- **balancing parsimony and accuracy (Zhang and Muhlenbein 1993 1994 1995; Blickle 1997)**
- **automatically defined features using genetic algorithms in conjunction with genetic programming (Andre 1994a)**
- **grammatical evolution (Conor Ryan and Michael O'Neill)**

## **GP TECHNIQUES — CONTINUED**

- **graphical program structures and neural programming (Teller and Veloso 1996, 1997; Teller 1998; Poli 1997a, 1997b)**
- **automatically defined macros (ADMs) for simultaneous evolution of programs and their control structures (Spector 1996)**
- **libraries (Koza 1990a; Koza and Rice 1991; Koza 1992a, section 6.5.4; Angeline and Pollack 1993, 1994; Angeline 1993, 1994; Kinnear 1994b)**
- **strong typing (Montana 1995; Montana and Czerwinski 1996; Janikow 1996; Yu and Clack 1997a) and constrained syntactic structures (Koza 1992a)**
- **explicit pointers (Andre 1994c)**
- **evolution of machine code (Nordin 1994, 1997) and linear genomes (Banzhaf, Nordin, Keller, and Francone 1998)**

# ARCHITECTURE-ALTERING OPERATIONS

## PROTEIN ALIGNMENT OF "A" AND "B" PROTEINS

First.protein MRIKFLVFLA VI~~C~~LFAHYAS ASGMGGDKKP KDAPKPKDAP KPKEVVKPV~~EA~~  
 Second.protein MRIKFLVFLA VI~~C~~LLAHYAS ASGMGGDKKP KDAPKPKDAP KPKEVVKPV~~EA~~

First.protein ESSEYEIEVI KHQEKTEKK EKE~~KK~~THVET KKEVKK~~KE~~KK QIPCSEK~~IL~~IK  
 Second.protein DSSEYEIEVI KHQEKTEKK EKE~~KK~~KAHVEI KKKIKN~~KE~~KK FVPCSEI~~IL~~IK

First.protein EKLD~~C~~ETKGV P~~A~~GYK~~A~~IF~~K~~F TEN~~E~~E-CDWT CDYEALPPPP GAKKDDK~~K~~~~AK~~  
 Second.protein EKLECEKNAT P-GYK~~A~~L~~F~~F~~F~~ KES~~S~~S~~F~~CEWE CDYEAI---P GAKKDEK~~K~~~~AK~~

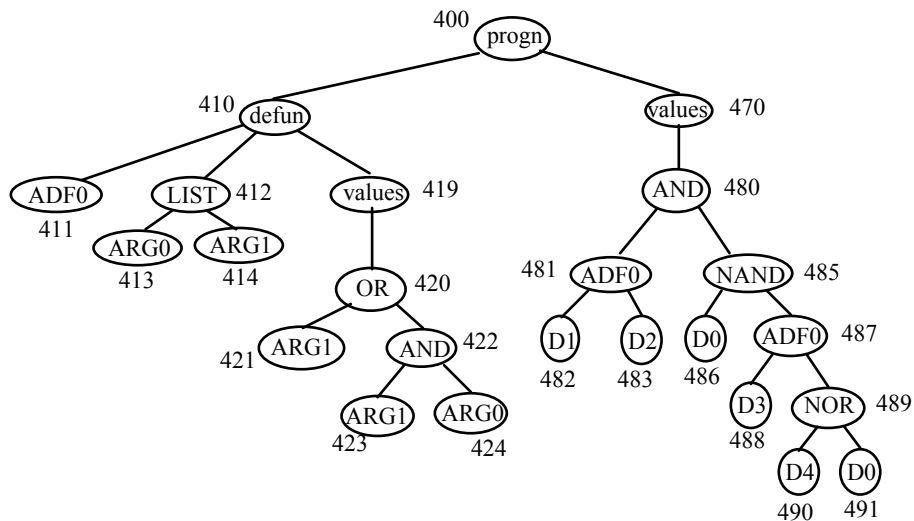
First.protein K~~I~~TVKV~~V~~KPPK EKPP~~K~~KLRKE CSGEKVIKFQ NCLVKIRGLI AFGDKTKN~~DE~~  
 Second.protein KVVKV~~V~~LKPPK EKPP~~K~~KPRKE CSGEKVIKFQ NCLVKIRGLI AFGDKTKN~~DE~~

First.protein KKFAKLVQGK QKKGAKKAKG G~~K~~KAAPKPG~~E~~ KPGPK~~-~~~~-~~Q ADKP---239  
 Second.protein KKFAKLVQGK QKKGAKKAKG G~~K~~KAEPKPG~~E~~ KPAPK~~E~~PGPK APKPVPK~~DAE~~

First.protein -~~-~~KDAKK 244  
 Second.protein K~~E~~KDAKK 253

# ARCHITECTURE-ALTERING OPERATIONS

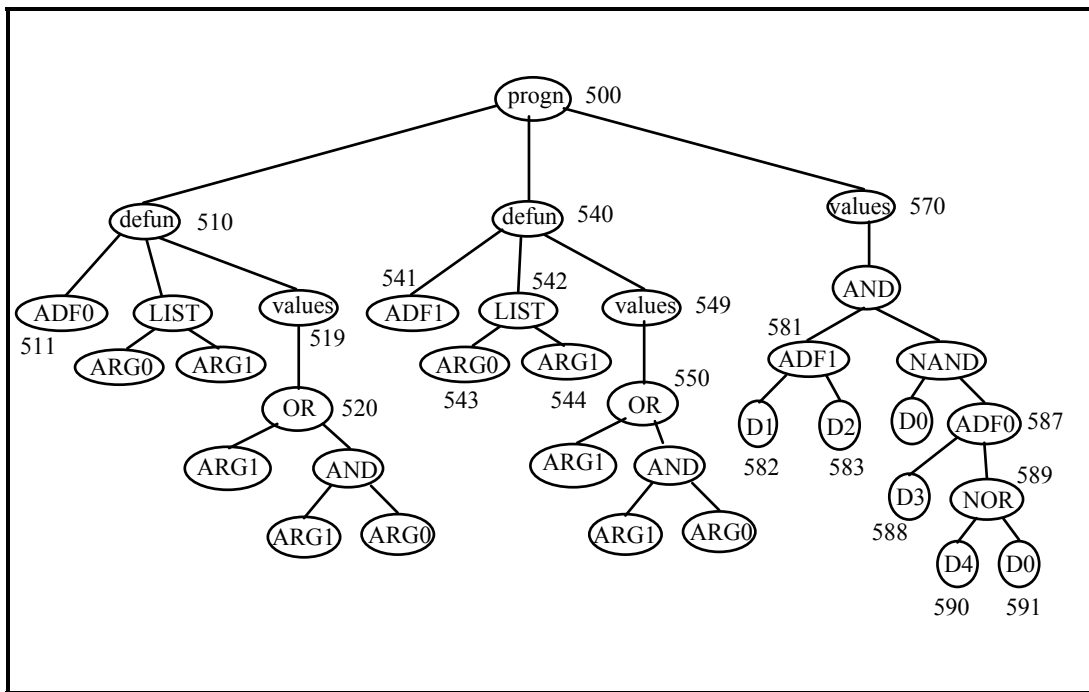
## PROGRAM WITH 1 TWO-ARGUMENT AUTOMATICALLY DEFINED FUNCTION (ADF0) AND 1 RESULT-PRODUCING BRANCH – ARGUMENT MAP OF {2}





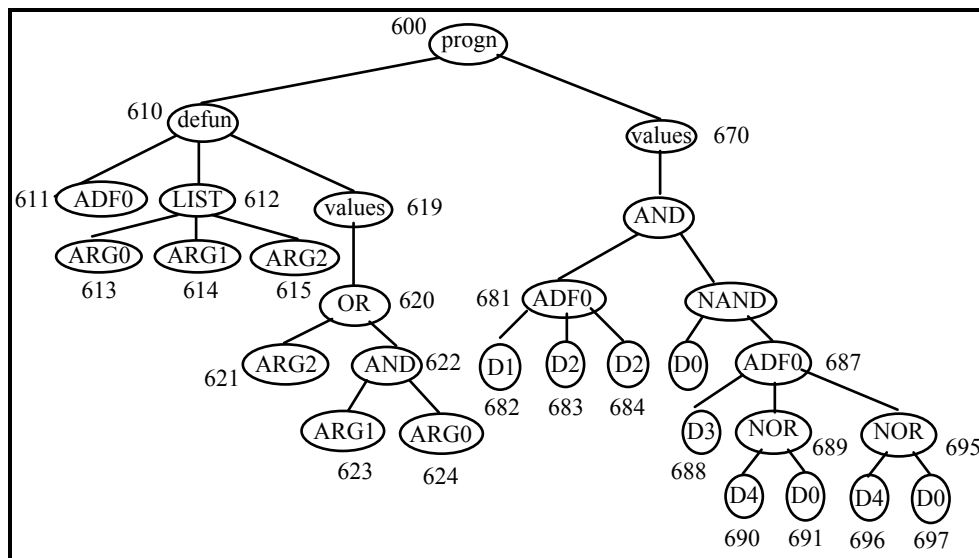
# ARCHITECTURE-ALTERING OPERATIONS

## PROGRAM WITH ARGUMENT MAP OF {2, 2} CREATED USING THE OPERATION OF BRANCH DUPLICATION



# ARCHITECTURE-ALTERING OPERATIONS

## PROGRAM WITH ARGUMENT MAP OF {3} CREATED USING THE OPERATION OF ARGUMENT DUPLICATION



# **ARCHITECTURE-ALTERING OPERATIONS**

## **SPECIALIZATION – REFINEMENT – CASE SPLITTING**

- **Branch duplication**
- **Argument duplication**
- **Branch creation**
- **Argument creation**

## **GENERALIZATION**

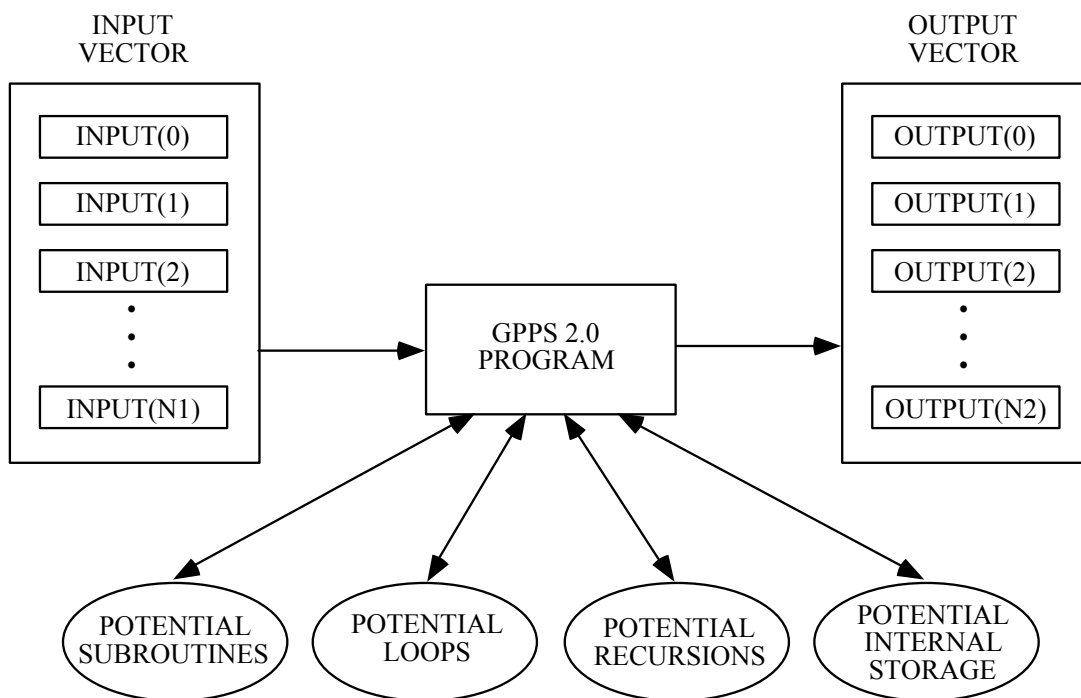
- **Branch deletion**
- **Argument deletion**

# **16 ATTRIBUTES OF A SYSTEM FOR AUTOMATICALLY CREATING COMPUTER PROGRAMS**

- 1 — Starts with "What needs to be done"**
- 2 — Tells us "How to do it"**
- 3 — Produces a computer program**
- 4 — Automatic determination of program size**
- 5 — Code reuse**
- 6 — Parameterized reuse**
- 7 — Internal storage**
- 8 — Iterations, loops, and recursions**
- 9 — Self-organization of hierarchies**
- 10 — Automatic determination of program architecture**
- 11 — Wide range of programming constructs**
- 12 — Well-defined**
- 13 — Problem-independent**
- 14 — Wide applicability**
- 15 — Scalable**
- 16 — Competitive with human-produced results**

# ARCHITECTURE-ALTERING OPERATIONS

## GENETIC PROGRAMMING PROBLEM SOLVER (GPPS) — VERSION 2.0



# **IMPLEMENTATION OF GP IN ASSEMBLY CODE – COMPILED GENETIC PROGRAMMING SYSTEM (NORDIN 1994)**

- Nordin, Peter. 1997. *Evolutionary Program Induction of Binary Machine Code and its Application*. Munster, Germany: Krehl Verlag.
- Opportunity to speed up GP that is done by slowly INTERPRETING GP program trees.  
Instead of interpreting the GP program tree, EXECUTE this sequence of assembly code.
- Can identify small set of primitive functions that is useful for large group of problems, such as +, -, \*, % and also use some conditional operations (IFLTE), some logical functions (AND, OR, XOR, XNOR) and perhaps others (e.g., SRL, SLL, SETHI from Sun 4).
- Then, generate random sequence of assembly code instructions at generation 0 from this small set of machine code instructions (referring to certain registers).
- If ADFs are involved, generate fixed header and footer of function and appropriate function call.
- Perform crossover possibly so as to preserve the integrity of subtrees.
- If ADFs are involved, perform crossover so as to preserve the integrity of the header and footer of function and the function call.

## DESIGN OF QUANTUM COMPUTER CIRCUITS USING GP (SPECTOR ET AL.)

- Spector, Lee, Barnum, Howard, and Bernstein, Herbert J. 1998. Genetic programming for quantum computers. In Koza, John R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, and Riolo, Rick. (editors). 1998. *Genetic Programming 1998: Proceedings of the Third Annual Conference*. San Francisco, CA: Morgan Kaufmann. Pages 365 - 373.
- Spector, Lee, Barnum, Howard, and Bernstein, Herbert J. 1999. Quantum computing applications of genetic programming. In Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter (editors). 1999. *Advances in Genetic Programming 3*. Cambridge, MA: The MIT Press. Pages 135-160.
- Spector, Lee, Barnum, Howard, Bernstein, Herbert J., and Swamy, N. 1999. Finding a better-than-classical quantum AND/OR algorithm using genetic programming. In IEEE. *Proceedings of 1999 Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Press. Pages 2239-2246.
- Barnum, H., Bernstein, H.J. and Spector, Lee. 2000. Quantum circuits for OR and AND of ORs. *Journal of Physics A: Mathematical and General*. 33 (45) 8047-8057. November 17, 2000).
- Spector, Lee, and Bernstein, Herbert J. 2003. Communication capacities of some quantum gates, discovered in part through genetic programming. In Shapiro, Jeffery H. and Hirota, Osamu (editors). *Proceedings of the Sixth International Conference on Quantum Communication, Measurement, and Computing*. Princeton, NJ: Rinton Press. Pages 500-503.

## **CELLULAR ENCODING (DEVELOPMENTAL GENETIC PROGRAMMING)**

- **Gruau, Frederic. 1992b. *Cellular Encoding of Genetic Neural Networks*. Technical report 92-21. Laboratoire de l'Informatique du Parallélisme. Ecole Normale Supérieure de Lyon. May 1992.**
- **Also: Gruau 1992a 1992b 1993 1994a 1994b; Gruau and Whitley 1993; Esparcia-Alcazar and Sharman 1997)**
- **Applied by Gruau and Whitley (1995) to 2-pole-balancing problem**
- **Applied by Gruau to six-legged walking creature**
- **Applied by Brave (1995, 1996) to Finite Automata**



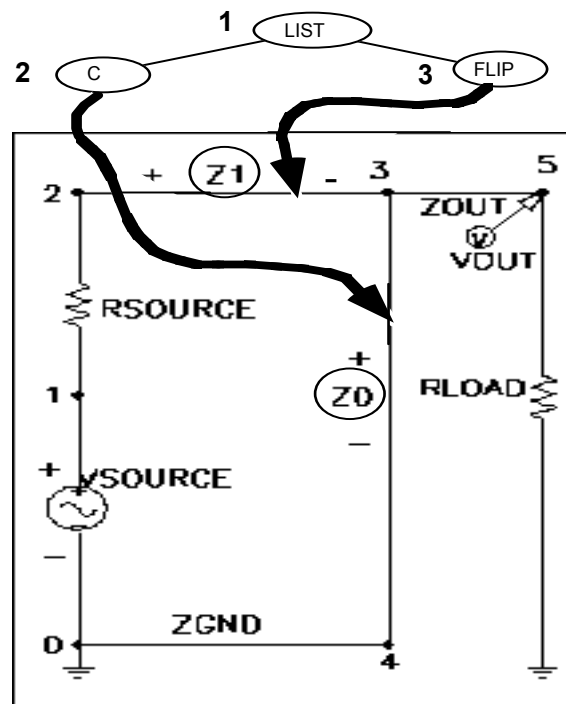
# **AUTOMATIC PARALLELIZATION OF SERIAL PROGRAMS USING GP**

- **Ryan, Conor. 1999. *Automatic Re-engineering of Software Using Genetic Programming*. Amsterdam: Kluwer Academic Publishers.**
- **Start with working serial computer program (embryo)**
- **GP program tree contains validity-preserving functions that modify the current program. That is, the functions in the program tree side-effect the current program.**
- **Execution of the complete GP program tree progressively modifies the current program**
- **Fitness is based on execution time on the parallel computer system**

# DEVELOPMENTAL GP

## THE INITIAL CIRCUIT

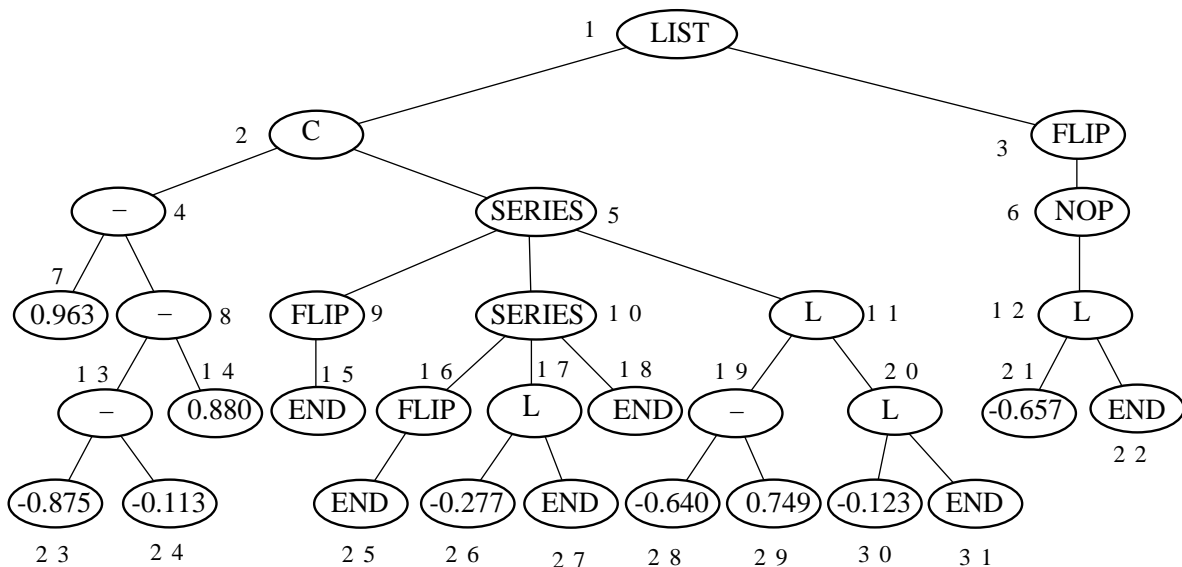
- Initial circuit consists of embryo and test fixture
- Embryo has modifiable wires (e.g., **Z0** AND **Z1**)
- Test fixture has input and output ports and usually has source resistor and load resistor. There are no modifiable wires (or modifiable components) in the test fixture.
- Circuit-constructing program trees consist of
  - Component-creating functions
  - Topology-modifying functions
  - Development-controlling functions
- Circuit-constructing program tree has one result-producing branch for each modifiable wire in embryo of the initial circuit



## DEVELOPMENTAL GP

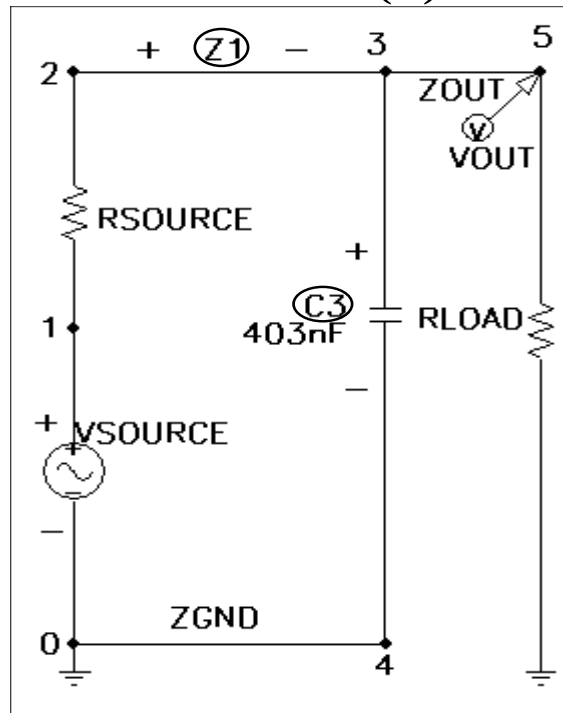
### DEVELOPMENT OF A CIRCUIT FROM A CIRCUIT-CONSTRUCTING PROGRAM TREE AND THE INITIAL CIRCUIT

```
(LIST (C (- 0.963 (- (- -0.875
-0.113) 0.880)) (series (flip
end) (series (flip end) (L -
0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip
(nop (L -0.657 end))))
```



## DEVELOPMENTAL GP

### RESULT OF THE c (2) FUNCTION

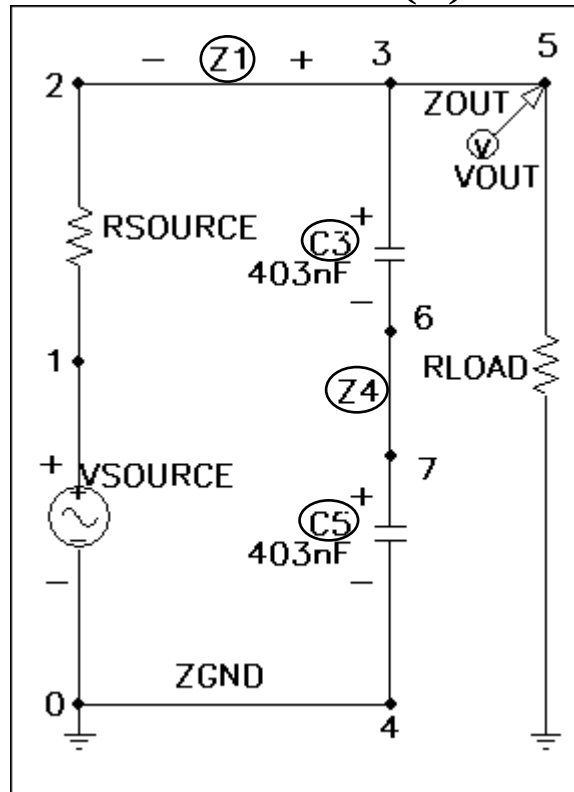


```
(LIST (C (- 0.963 (- (- -0.875
-0.113) 0.880)) (series (flip
end) (series (flip end) (L -
0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip
(nop (L -0.657 end))))
```

NOTE: Interpretation of arithmetic value

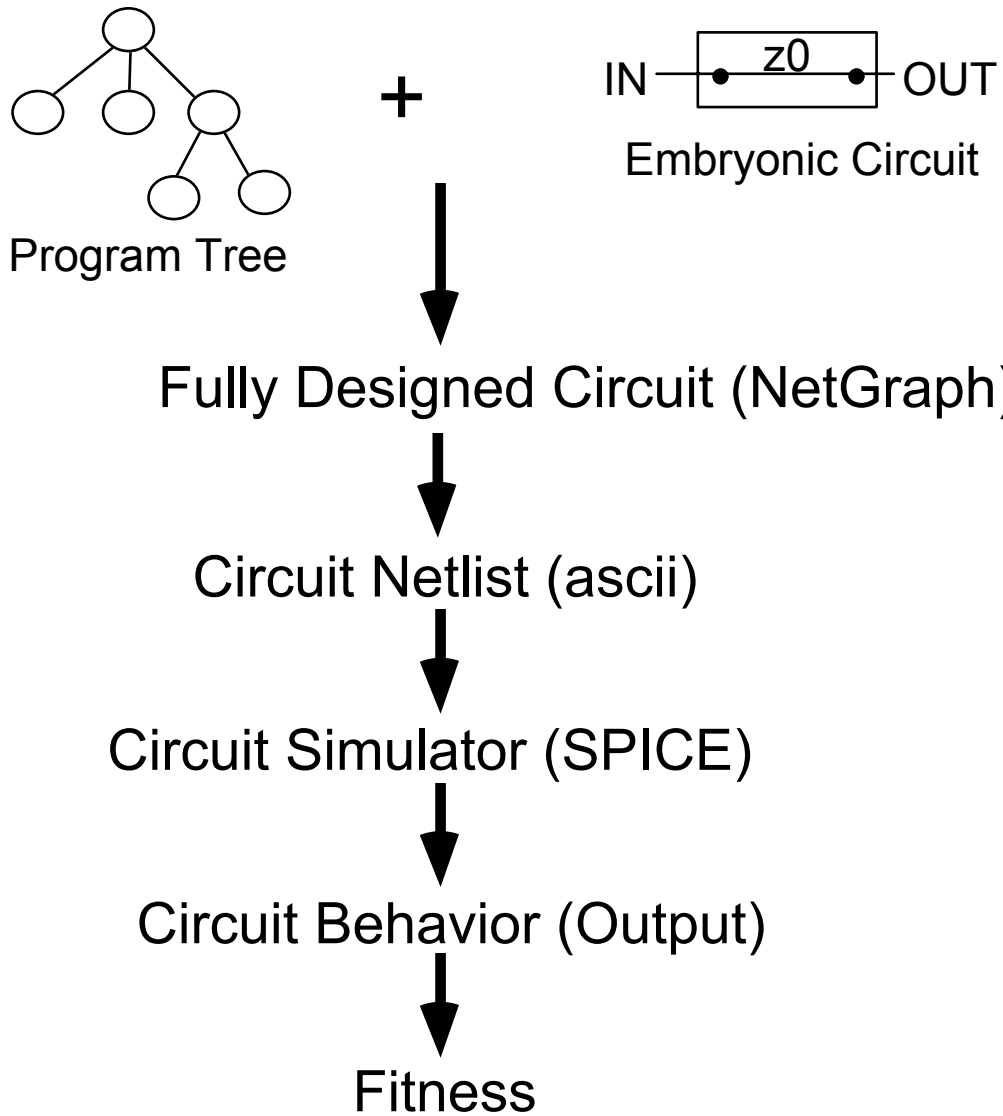
## DEVELOPMENTAL GP

### RESULT OF SERIES (5) FUNCTION

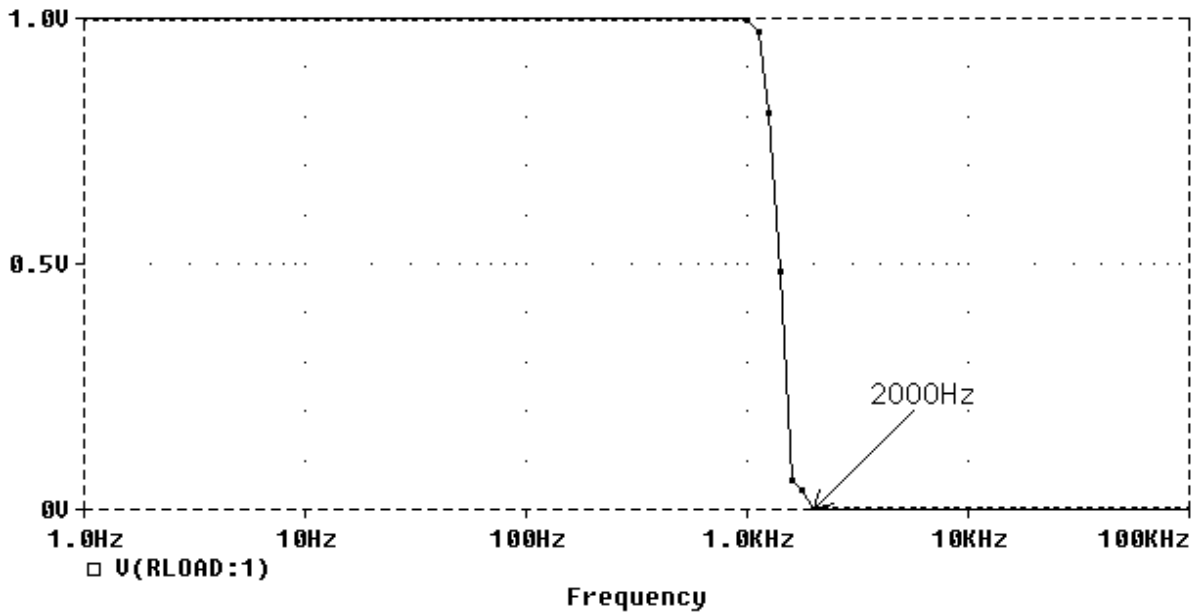


```
(LIST (C (- 0.963 (- (- -0.875
-0.113) 0.880)) (series (flip
end) (series (flip end) (L -
0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip
(nop (L -0.657 end))))
```

# EVALUATION OF FITNESS OF A CIRCUIT



## BEHAVIOR OF A LOWPASS FILTER VIEWED IN THE FREQUENCY DOMAIN



- Examine circuit's behavior for each of 101 frequency values chosen over five decades of frequency (from 1 Hz to 100,000 Hz) with each decade divided into 20 parts (using a logarithmic scale). The fitness measure
  - does not penalize ideal values
  - slightly penalizes acceptable deviations
  - heavily penalizes unacceptable deviations
- Fitness is sum  $F(t) = \sum_{i=0}^{100} [W(f_i)d(f_i)]$ 
  - $f(i)$  is the frequency of fitness case  $i$
  - $d(x)$  is the difference between the target and observed values at frequency of fitness case  $i$
  - $W(y,x)$  is the weighting at frequency  $x$

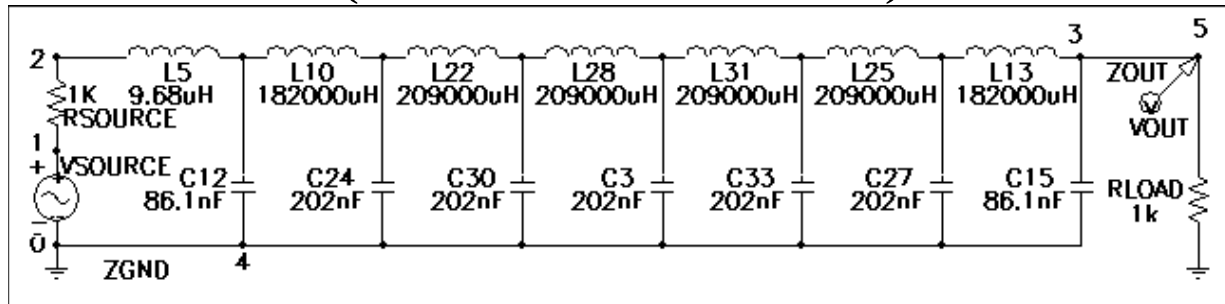
## TABLEAU — LOWPASS FILTER (WITHOUT ADFs OR ARCHITECTURE- ALTERING OPERATIONS)

<b>Objective:</b>	Design a lowpass filter composed of inductors and capacitors with a passband below 1,000 Hz, a stopband above 2,000 Hz, a maximum allowable passband deviation of 30 millivolts, and a maximum allowable stopband deviation of 1 millivolt.
<b>Test fixture and embryo:</b>	One-input, one-output initial circuit with a source resistor, load resistor, and two modifiable wires.
<b>Program architecture:</b>	Two result-producing branches, RPB0 and RPB1 (i.e., one RPB per modifiable wire in the embryo).
<b>Initial function set for the result-producing branches:</b>	<p>For construction-continuing subtrees:  <math>F_{\text{ccs-rpb-initial}} = \{C, L, \text{SERIES}, \text{PARALLEL0}, \text{FLIP}, \text{NOP}, \text{TWO\_GROUND}, \text{TWO\_VIA0}, \text{TWO\_VIA1}, \text{TWO\_VIA2}, \text{TWO\_VIA3}, \text{TWO\_VIA4}, \text{TWO\_VIA5}, \text{TWO\_VIA6}, \text{TWO\_VIA7}\}.</math></p> <p>For arithmetic-performing subtrees:  <math>F_{\text{aps}} = \{+, -\}.</math></p>
<b>Initial terminal set for the result-producing branches:</b>	<p>For construction-continuing subtrees:  <math>T_{\text{ccs-rpb-initial}} = \{\text{END}\}.</math></p> <p>For arithmetic-performing subtrees:  <math>T_{\text{aps}} = \{\leftarrow\text{smaller-reals}\}.</math></p>



<b>Fitness cases:</b>	<b>101 frequency values in an interval of five decades of frequency values between 1 Hz and 100,000 Hz.</b>
<b>Raw fitness:</b>	<b>Fitness is the sum, over the 101 sampled frequencies (fitness cases), of the absolute weighted deviation between the actual value of the output voltage that is produced by the circuit at the probe point and the target value for voltage. The weighting penalizes unacceptable output voltages much more heavily than deviating, but acceptable, voltages.</b>
<b>Standardized fitness:</b>	<b>Same as raw fitness.</b>
<b>Hits:</b>	<b>The number of hits is defined as the number of fitness cases (out of 101) for which the voltage is acceptable or ideal or that lie in the "don't care" band.</b>
<b>Wrapper:</b>	<b>None.</b>
<b>Parameters:</b>	<b><math>M = 1,000</math> to <math>320,000</math>. <math>G = 1,001</math>. <math>Q = 1,000</math>. <math>D = 64</math>. <math>B = 2\%</math>. <math>N_{rpb} = 2</math>. <math>S_{rpb} = 200</math>.</b>
<b>Result designation:</b>	<b>Best-so-far pace-setting individual.</b>
<b>Success predicate:</b>	<b>A program scores the maximum number (101) of hits.</b>

## EVOLVED CAMPBELL FILTER (7-RUNG LADDER)

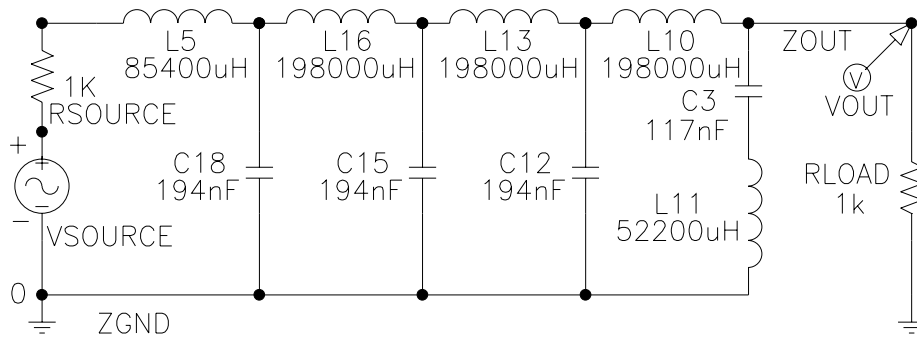


- This genetically evolved circuit infringes on U. S. patent 1,227,113 issued to George Campbell of American Telephone and Telegraph in 1917 (claim 2):

An electric wave filter consisting of a connecting line of negligible attenuation composed of a plurality of sections, each section including a capacity element and an inductance element, one of said elements of each section being in series with the line and the other in shunt across the line, said capacity and inductance elements having precomputed values dependent upon the upper limiting frequency and the lower limiting frequency of a range of frequencies it is desired to transmit without attenuation, the values of said capacity and inductance elements being so proportioned that the structure transmits with practically negligible attenuation sinusoidal currents of all frequencies lying between said two limiting frequencies, while attenuating and approximately extinguishing currents of neighboring frequencies lying outside of said limiting frequencies."

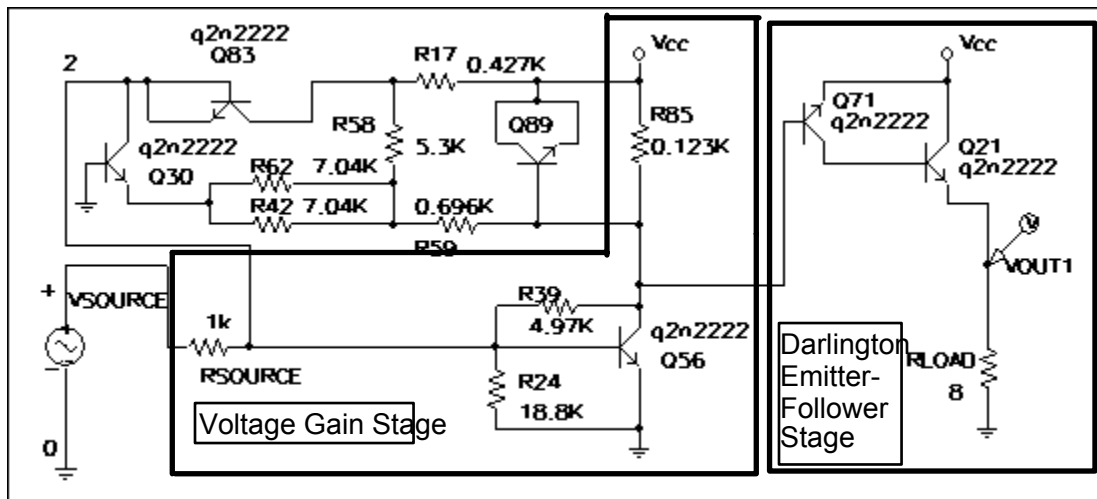
## EVOLVED ZOBEL FILTER

- Infringes on U. S. patent 1,538,964 issued in 1925 to Otto Zobel of American Telephone and Telegraph Company for an “*M*-derived half section” used in conjunction with one or more “constant *K*” sections.
- One *M*-derived half section (C2 and L11)
- Cascade of three symmetric T-sections



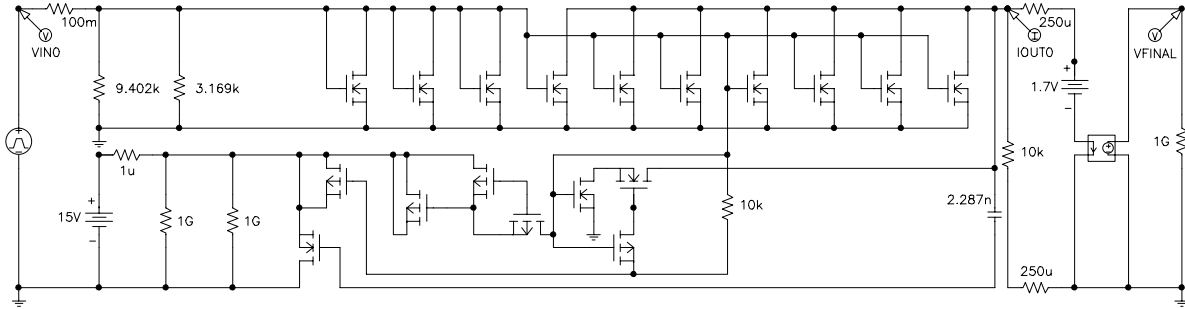
# GENETICALLY EVOLVED 10 DB AMPLIFIER FROM GENERATION 45

## SHOWING THE VOLTAGE GAIN STAGE AND DARLINGTON EMITTER FOLLOWER SECTION



# POST-2000 PATENTED INVENTIONS

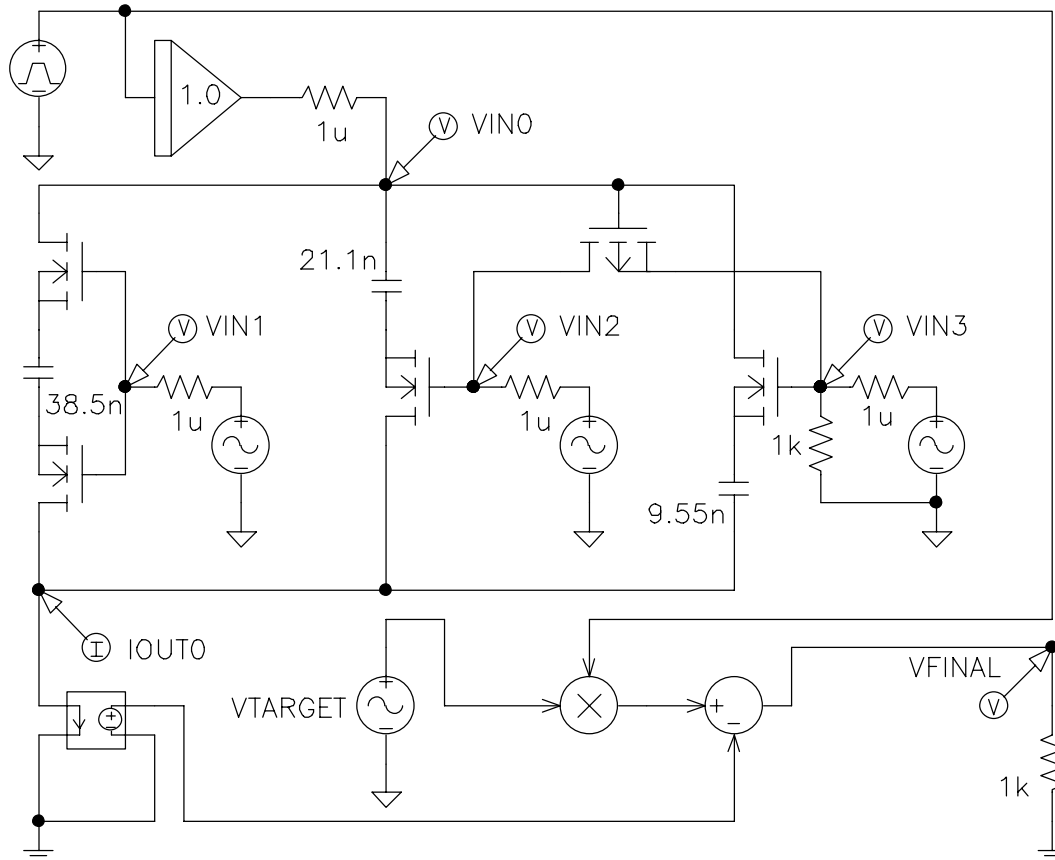
## HIGH CURRENT LOAD CIRCUIT BEST-OF-RUN FROM GENERATION 114



# POST-2000 PATENTED INVENTIONS

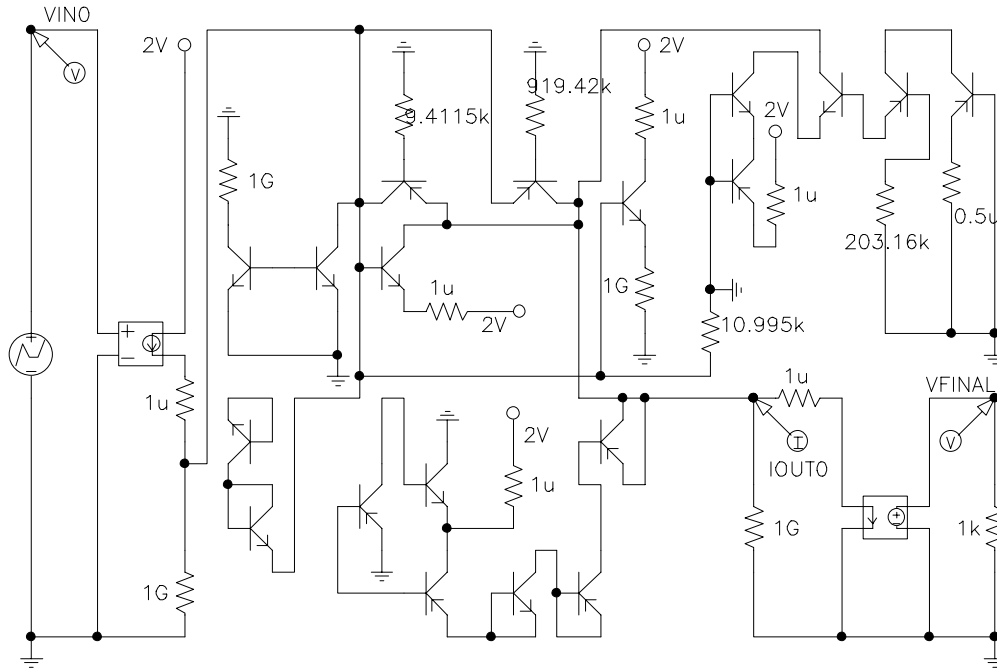
## REGISTER-CONTROLLED CAPACITOR CIRCUIT

### SMALLEST COMPLIANT FROM GENERATION 98



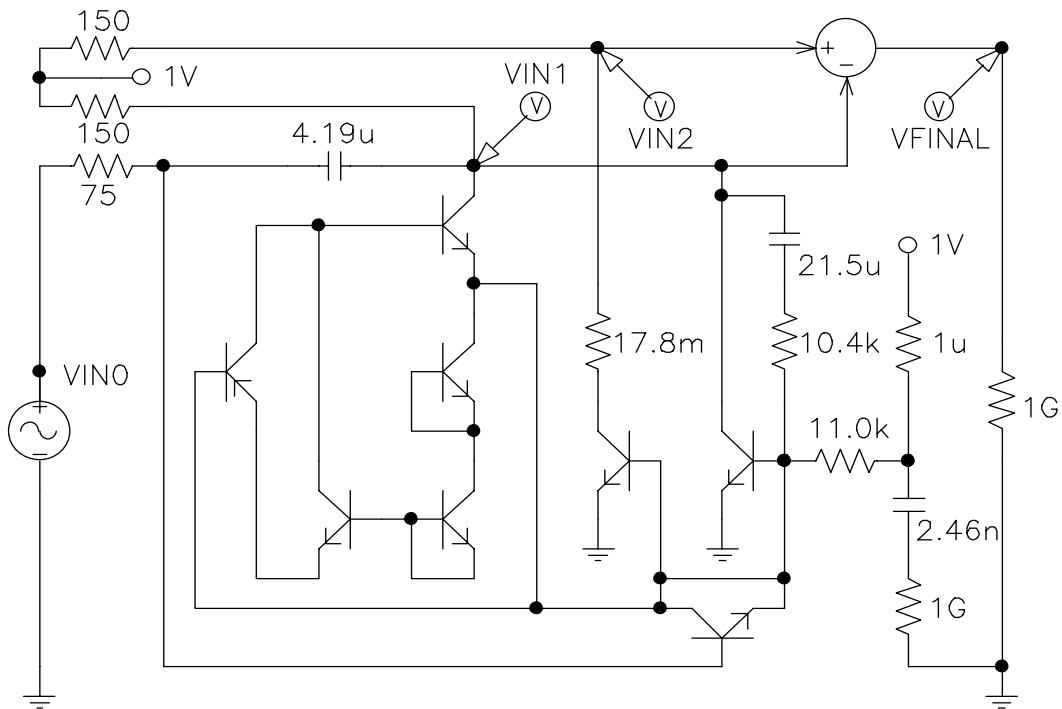
# POST-2000 PATENTED INVENTIONS

## LOW-VOLTAGE CUBIC SIGNAL GENERATION CIRCUIT BEST-OF-RUN FROM GENERATION 182



# POST-2000 PATENTED INVENTIONS

## LOW-VOLTAGE BALUN CIRCUIT BEST EVOLVED FROM GENERATION 84



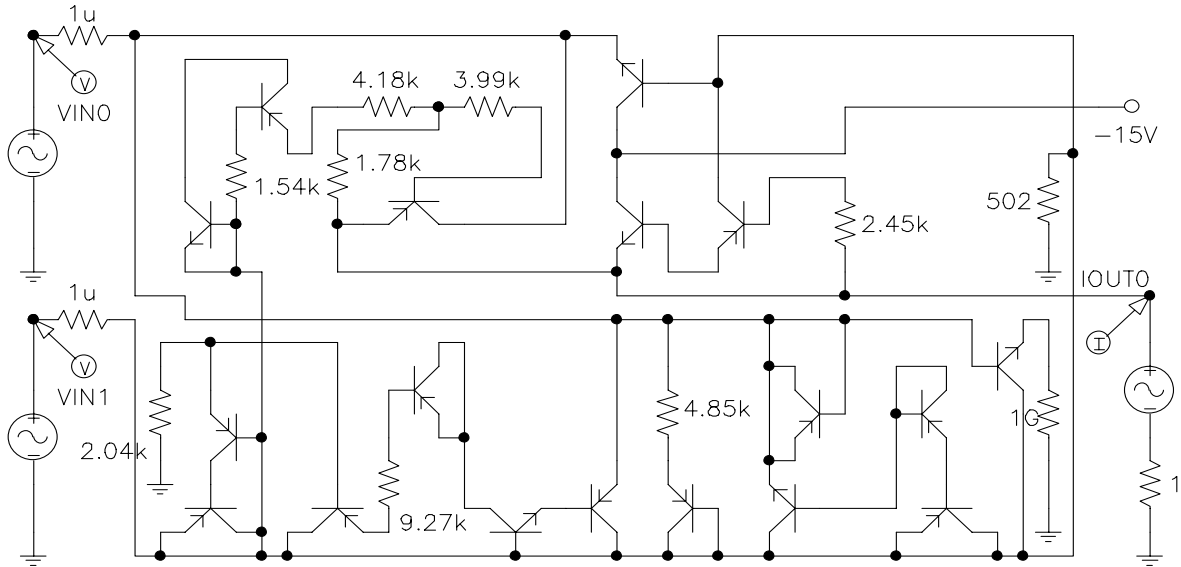


**POST-2000 PATENTED INVENTIONS**

**VOLTAGE-CURRENT-CONVERSION**

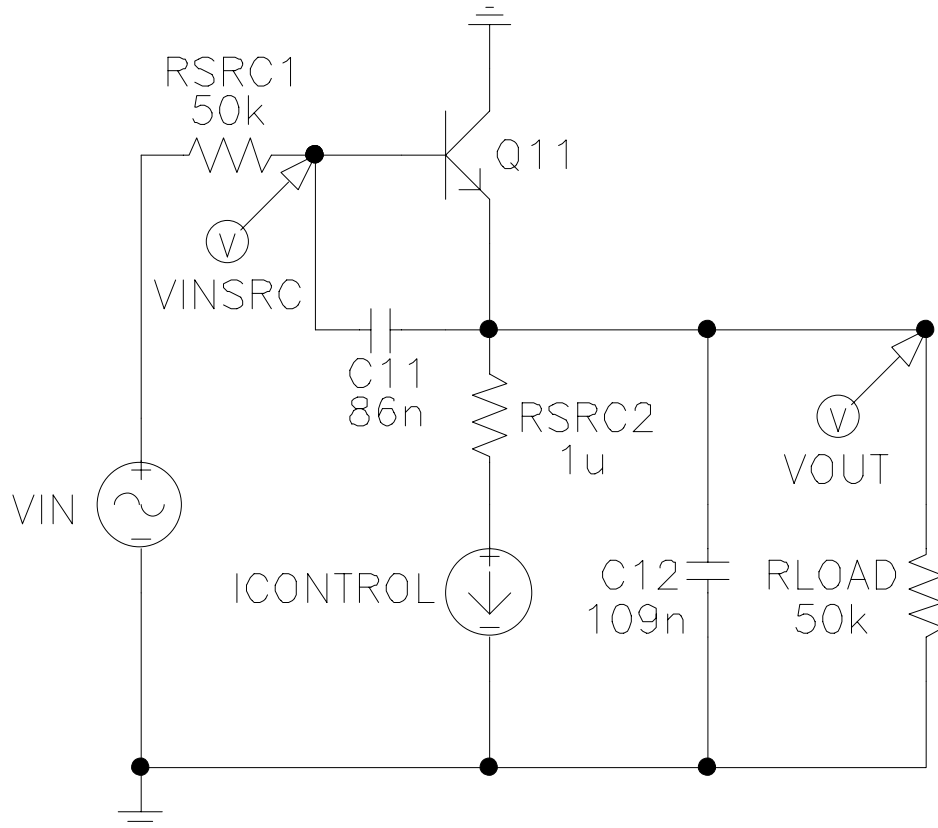
**CIRCUIT**

**BEST-OF-RUN FROM GENERATION 109**



# POST-2000 PATENTED INVENTIONS

## TUNABLE INTEGRATED ACTIVE FILTER — GENERATION 50



## 21 PREVIOUSLY PATENTED INVENTIONS REINVENTED BY GP

	<b>Invention</b>	<b>Date</b>	<b>Inventor</b>	<b>Place</b>	<b>Patent</b>
<b>1</b>	<b>Darlington emitter-follower section</b>	<b>1953</b>	<b>Sidney Darlington</b>	<b>Bell Telephone Laboratories</b>	<b>2,663,806</b>
<b>2</b>	<b>Ladder filter</b>	<b>1917</b>	<b>George Campbell</b>	<b>American Telephone and Telegraph</b>	<b>1,227,113</b>
<b>3</b>	<b>Crossover filter</b>	<b>1925</b>	<b>Otto Julius Zobel</b>	<b>American Telephone and Telegraph</b>	<b>1,538,964</b>
<b>4</b>	<b>“M-derived half section” filter</b>	<b>1925</b>	<b>Otto Julius Zobel</b>	<b>American Telephone and Telegraph</b>	<b>1,538,964</b>
<b>5</b>	<b>Cauer (elliptic) topology for filters</b>	<b>1934–1936</b>	<b>Wilhelm Cauer</b>	<b>University of Gottingen</b>	<b>1,958,742, 1,989,545</b>
<b>6</b>	<b>Sorting network</b>	<b>1962</b>	<b>Daniel G. O’Connor and Raymond J. Nelson</b>	<b>General Precision, Inc.</b>	<b>3,029,413</b>
<b>7</b>	<b>Computational circuits</b>	<b>See text</b>	<b>See text</b>	<b>See text</b>	<b>See text</b>
<b>8</b>	<b>Electronic thermometer</b>	<b>See text</b>	<b>See text</b>	<b>See text</b>	<b>See text</b>
<b>9</b>	<b>Voltage reference circuit</b>	<b>See text</b>	<b>See text</b>	<b>See text</b>	<b>See text</b>
<b>10</b>	<b>60 dB and 96 dB amplifiers</b>	<b>See text</b>	<b>See text</b>	<b>See text</b>	<b>See text</b>
<b>11</b>	<b>Second-derivative controller</b>	<b>1942</b>	<b>Harry Jones</b>	<b>Brown Instrument Company</b>	<b>2,282,726</b>
<b>12</b>	<b>Philbrick circuit</b>	<b>1956</b>	<b>George Philbrick</b>	<b>George A. Philbrick Researches</b>	<b>2,730,679</b>
<b>13</b>	<b>NAND circuit</b>	<b>1971</b>	<b>David H. Chung and Bill H.</b>	<b>Texas Instruments Incorporated</b>	<b>3,560,760</b>

			<b>Terrell</b>		
<b>14</b>	<b>PID (proportional, integrative, and derivative) controller</b>	<b>1939</b>	<b>Albert Callender and Allan Stevenson</b>	<b>Imperial Chemical Limited</b>	<b>2,175,985</b>
<b>15</b>	<b>Negative feedback</b>	<b>1937</b>	<b>Harold S. Black</b>	<b>American Telephone and Telegraph</b>	<b>2,102,670, 2,102,671</b>
<b>16</b>	<b>Low-voltage balun circuit</b>	<b>2001</b>	<b>Sang Gug Lee</b>	<b>Information and Communications University</b>	<b>6,265,908</b>
<b>17</b>	<b>Mixed analog-digital variable capacitor circuit</b>	<b>2000</b>	<b>Turgut Sefket Aytur</b>	<b>Lucent Technologies Inc.</b>	<b>6,013,958</b>
<b>18</b>	<b>High-current load circuit</b>	<b>2001</b>	<b>Timothy Daun-Lindberg and Michael Miller</b>	<b>International Business Machines Corporation</b>	<b>6,211,726</b>
<b>19</b>	<b>Voltage-current conversion circuit</b>	<b>2000</b>	<b>Akira Ikeuchi and Naoshi Tokuda</b>	<b>Mitsumi Electric Co., Ltd.</b>	<b>6,166,529</b>
<b>20</b>	<b>Cubic function generator</b>	<b>2000</b>	<b>Stefano Cipriani and Anthony A. Takeshian</b>	<b>Conexant Systems, Inc.</b>	<b>6,160,427</b>
<b>21</b>	<b>Tunable integrated active filter</b>	<b>2001</b>	<b>Robert Irvine and Bernd Kolb</b>	<b>Infineon Technologies AG</b>	<b>6,225,859</b>

## **2 PATENTABLE INVENTIONS CREATED BY GENETIC PROGRAMMING**

	<b>Claimed invention</b>	<b>Date of patent application</b>	<b>Inventors</b>
<b>1</b>	<b>Improved general-purpose tuning rules for a PID controller</b>	<b>July 12, 2002</b>	<b>Martin A. Keane, John R. Koza, and Matthew J. Streeter</b>
<b>2</b>	<b>Improved general-purpose non-PID</b>	<b>July 12, 2002</b>	<b>Martin A. Keane, John R. Koza, and Matthew J. Streeter</b>

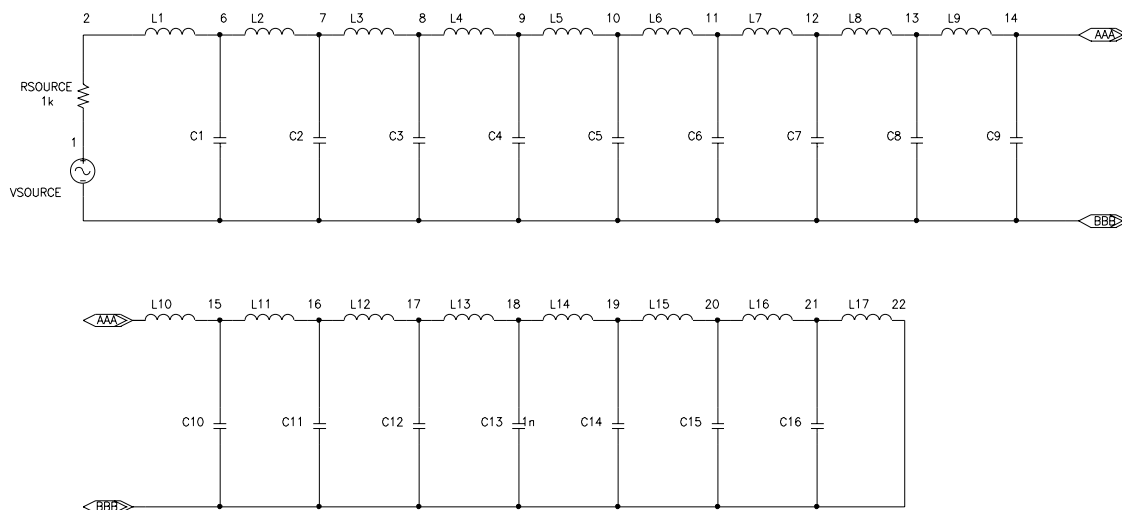
	<b>controllers</b>		
--	--------------------	--	--

# NOVELTY-DRIVEN EVOLUTION

## EXAMPLE OF LOWPASS FILTER

- Two factors in fitness measure
  - Circuit's behavior in the frequency domain
  - Largest number of nodes and edges (circuit components) of a subgraph of the given circuit that is isomorphic to a subgraph of a template representing the prior art. Graph isomorphism algorithm with the cost function being based on the number of shared nodes and edges (instead of just the number of nodes).

### PRIOR ART TEMPLATE

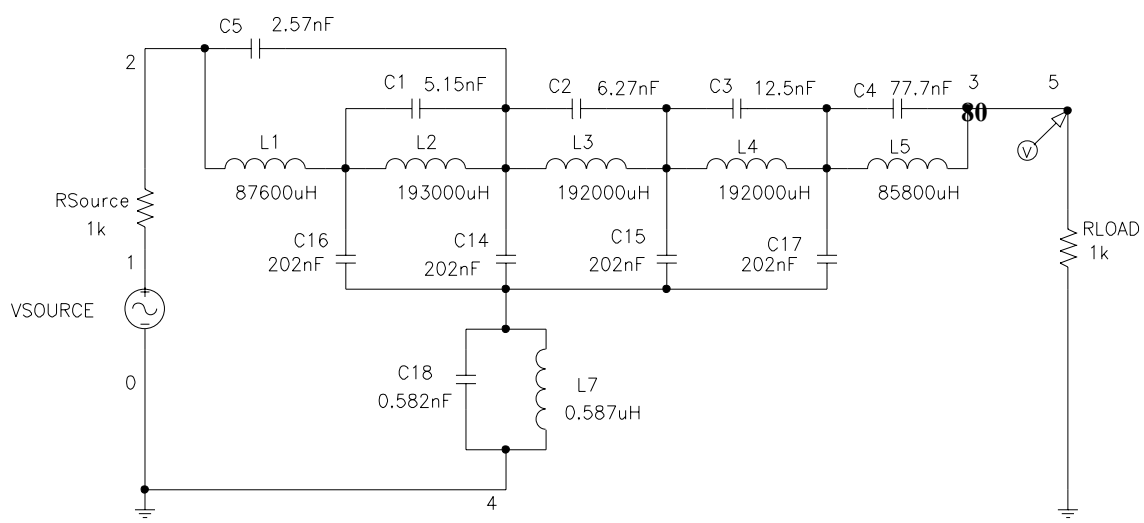


## NOVELTY-DRIVEN EVOLUTION — CONTINUED

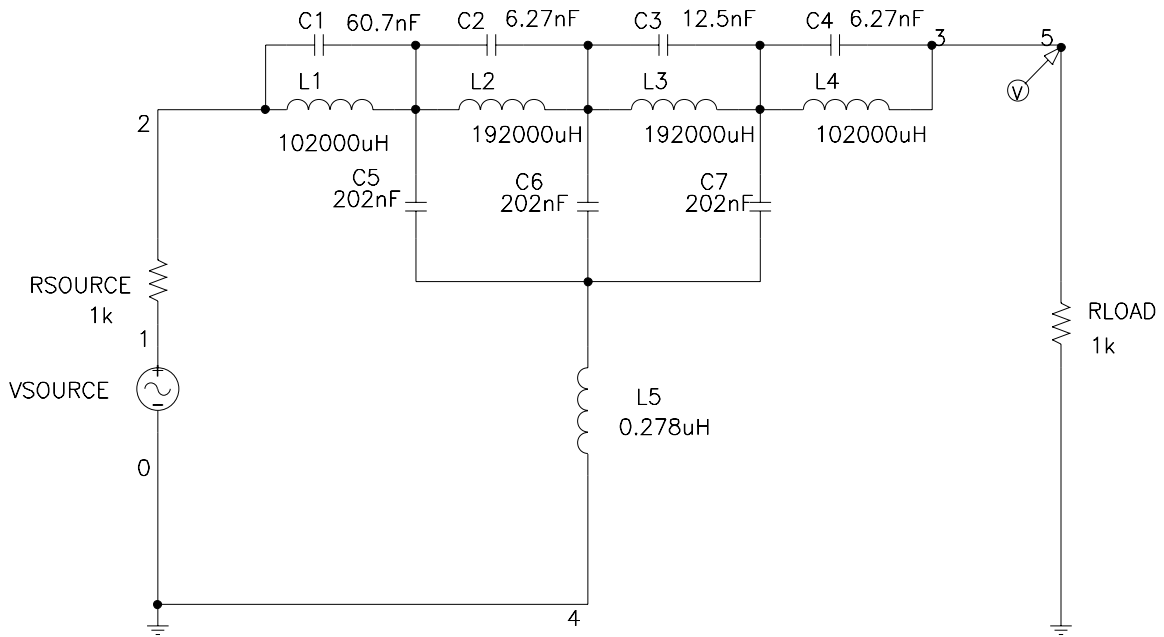
- For circuits not scoring the maximum number (101) of hits, the fitness of a circuit is the product of the two factors.
- For circuits scoring 101 hits (100%-compliant individuals), fitness is the number of shared nodes and edges divided by 10,000.

### FITNESS OF EIGHT 100%-COMPLIANT CIRCUITS

Solution	Frequency factor	Isomorphism factor	Fitness
1	0.051039	7	0.357273
2	0.117093	7	0.819651
3	0.103064	7	0.721448
4	0.161101	7	1.127707
5	0.044382	13	0.044382
6	0.133877	7	0.937139
7	0.059993	5	0.299965
8	0.062345	11	0.685795



## SOLUTION NO. 1

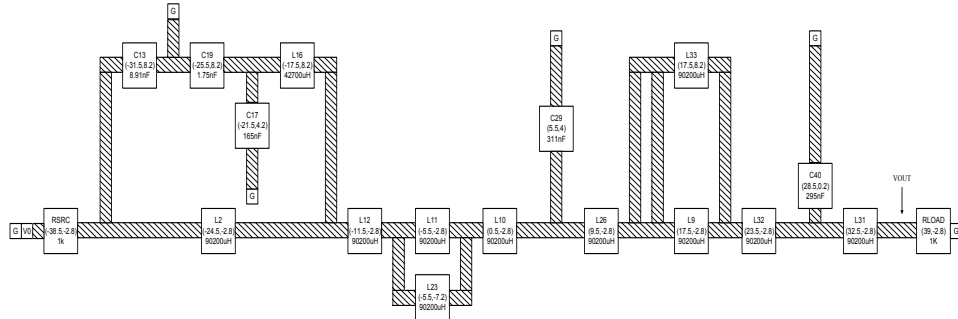


## SOLUTION NO. 5

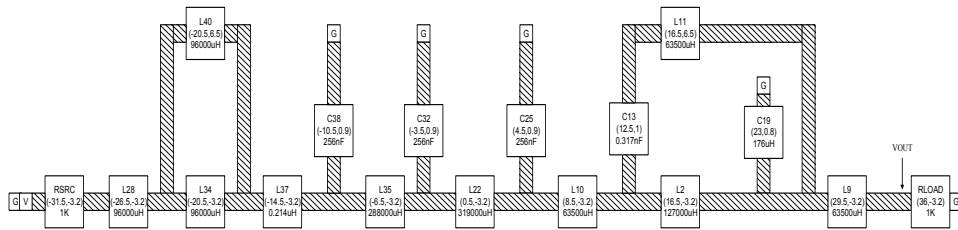


# LAYOUT — LOWPASS FILTER 100%-COMPLIANT CIRCUITS

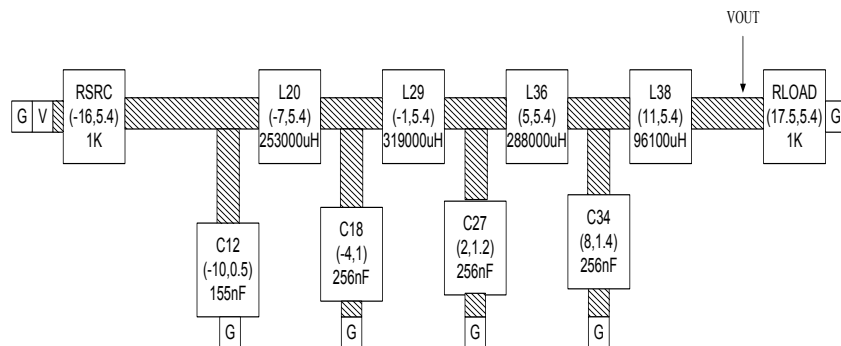
## GENERATION 25 WITH 5 CAPACITORS AND 11 INDUCTORS — AREA OF 1775.2



## GENERATION 30 WITH 10 INDUCTORS AND 5 CAPACITORS — AREA OF 950.3



## BEST-OF-RUN CIRCUIT OF GENERATION 138 WITH 4 INDUCTORS AND 4 CAPACITORS — AREA OF 359.4

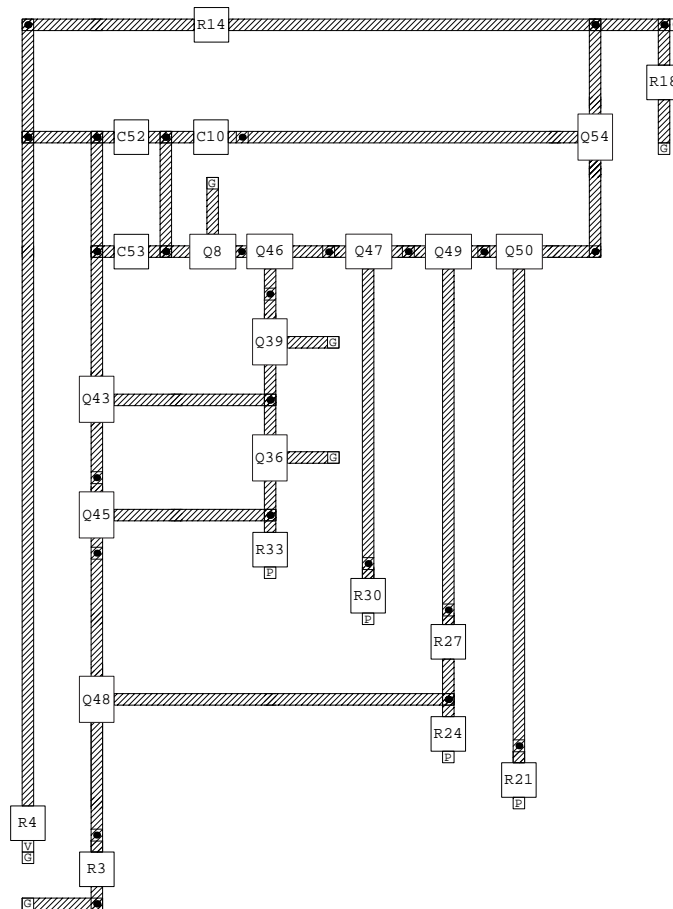


# LAYOUT — 60 DB AMPLIFIER (USING TRANSISTORS)

## COMPARISON

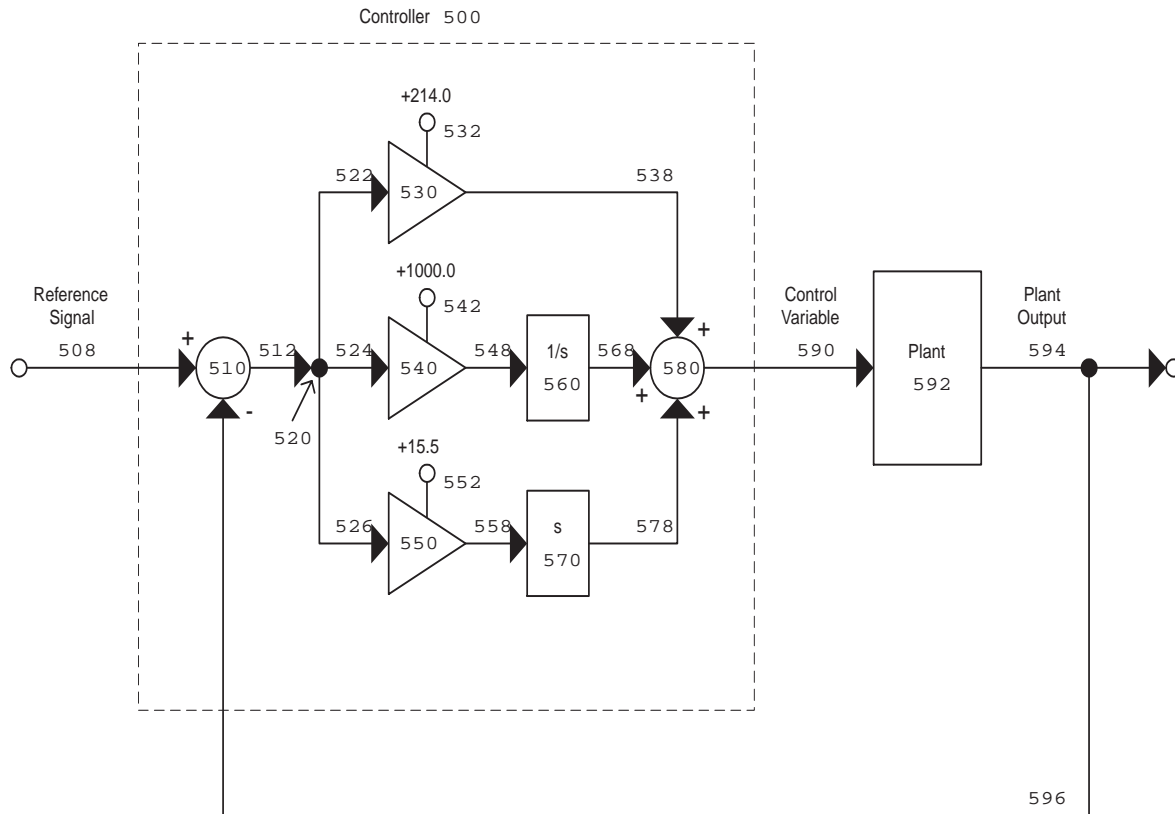
Gen	Components	Area	Four penalties	Fitness
65	27	8,234	33.034348	33.042583
101	19	4,751	0.061965	0.004751

## BEST-OF-RUN CIRCUIT FROM GENERATION 101



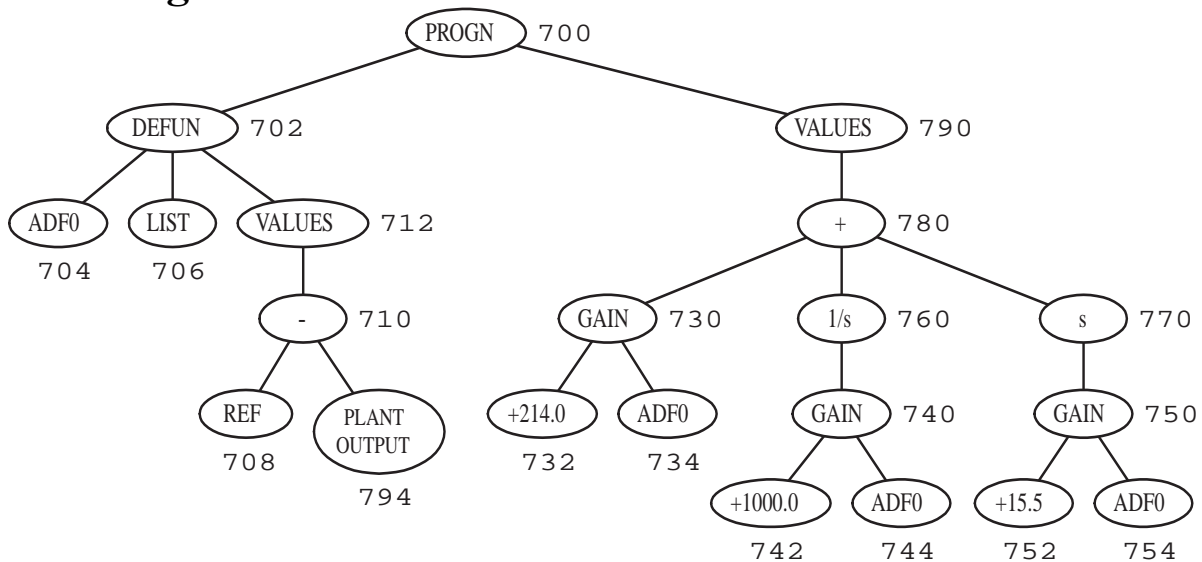
# PID CONTROLLER

**Block diagram of a plant and a PID controller composed of proportional (P), integrative (I), and derivative (D) blocks**

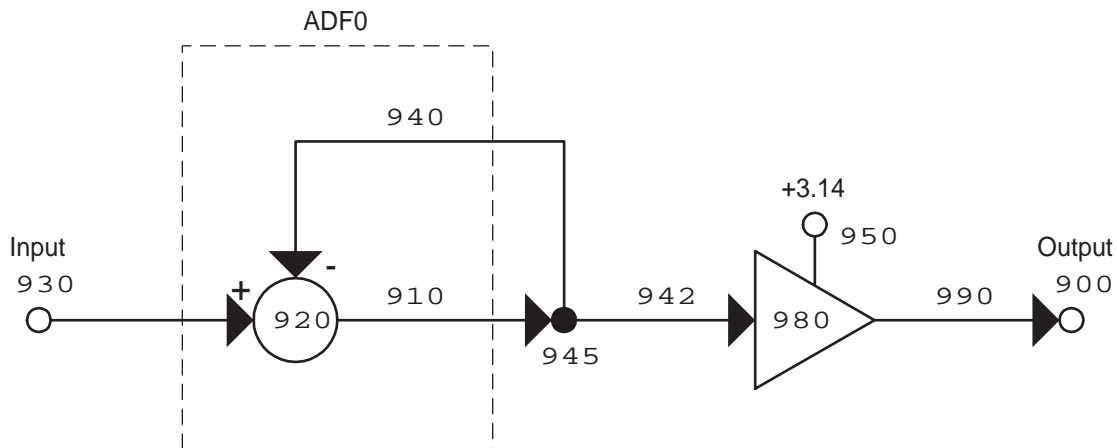


# PROGRAM TREE REPRESENTATION FOR PID CONTROLLER

- ADF can be used for reuse.
- Automatically defined function ADF0 takes the difference between the reference signal and the plant output and makes this difference available to three points in the result-producing branch



- ADF can be used for internal feedback



## FUNCTION SET AND TERMINAL SET FOR TWO-LAG PLANT PROBLEM

- The function set, **F** (for every part of the result-producing branch and any automatically defined functions except the arithmetic-performing subtrees) is

$$\mathbf{F} = \{ \text{GAIN, INVERTER, LEAD, LAG, LAG2, DIFFERENTIAL\_INPUT\_INTEGRATOR, DIFFERENTIATOR, ADD\_SIGNAL, SUB\_SIGNAL, ADD\_3\_SIGNAL, ADF0, ADF1, ADF2, ADF3, ADF4} \}$$

- The terminal set, **T**, (for every part of the result-producing branch and any automatically defined functions except the arithmetic-performing subtrees) is

$$\mathbf{T} = \{ \text{REFERENCE\_SIGNAL, CONTROLLER\_OUTPUT, PLANT\_OUTPUT, CONSTANT\_0} \}$$

## ARITHMETIC-PERFORMING SUBTREES FOR THE TWO-LAG PLANT PROBLEM

- Signal processing blocks such as GAIN, LEAD, LAG, and LAG2 possess numerical parameter(s)
- Parameter values can be established by an arithmetic-performing subtree
- A constrained syntactic structure enforces a different function and terminal set for the arithmetic-performing subtrees (as opposed to all other parts of the program tree).
- Terminal set,  $T_{aps}$ , for the arithmetic-performing subtrees

$$T_{aps} = \{\mathcal{R}\}$$

where  $\mathcal{R}$  denotes constant numerical terminals in the range from -1.0 to +1.0

- Function set,  $F_{aps}$ , for the arithmetic-performing subtrees
- $$F_{aps} = \{\text{ADD\_NUMERIC, SUB\_NUMERIC}\}$$

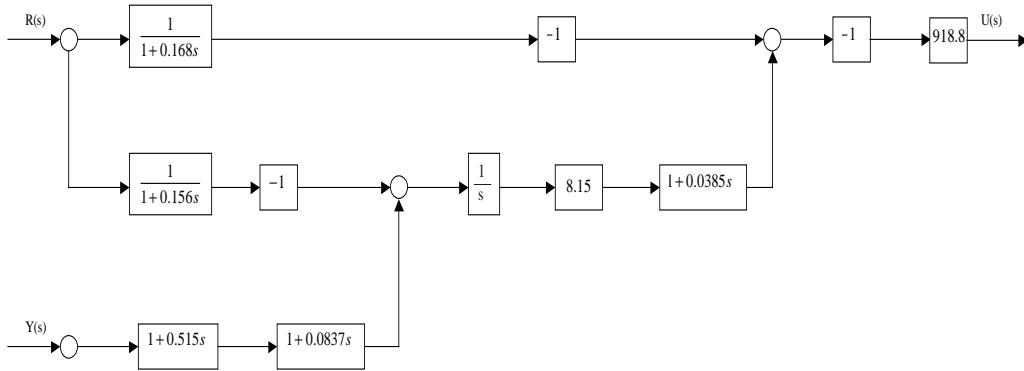
## FITNESS MEASURE FOR TWO-LAG PLANT

- 10-element fitness measure
- The first eight elements of the fitness measure represent the eight choices of a particular one of two different values of the plant's internal gain,  $K$  (1.0 and 2.0), in conjunction with a particular one of two different values of the plant's time constant  $\tau$  (0.5 and 1.0), in conjunction with a particular one of two different values for the height of the reference signal. The two reference signals are step functions that rise from 0 to 1 volts (or 1 microvolts) at  $t = 100$  milliseconds.
- For each of these eight fitness cases, a transient analysis is performed in the time domain using the SPICE simulator. The contribution to fitness for each of these eight elements is

$$\int_{t=0}^{9.6} t |e(t)| A(e(t)) B dt$$

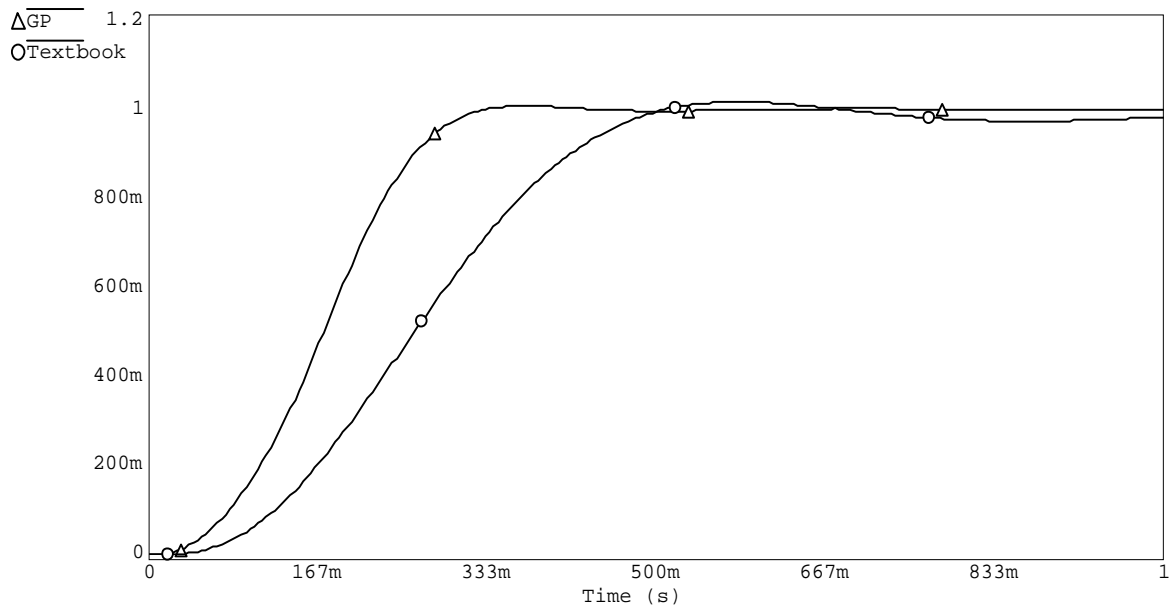
- $e(t)$  is difference between plant output and reference signal.
- Multiplication by  $B$  ( $10^6$ . or 1) makes both reference signals equally influential.
- Additional weighting function,  $A$ , heavily penalizes non-compliant amounts of overshoot.  $A$  weights all variations up to 2% above the reference signal by 1.0, but others by 10.0.
- The 9<sup>th</sup> element of the fitness measure exposes the controller to an extreme spiked reference signal.
- The 10<sup>th</sup> element constrains the frequency of the control variable so as to avoid extreme high frequencies.

# BEST-OF-RUN GENETICALLY EVOLVED CONTROLLER FROM GENERATION 32 FOR THE TWO-LAG PLANT

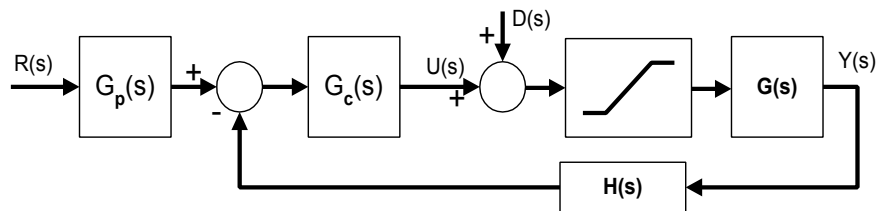




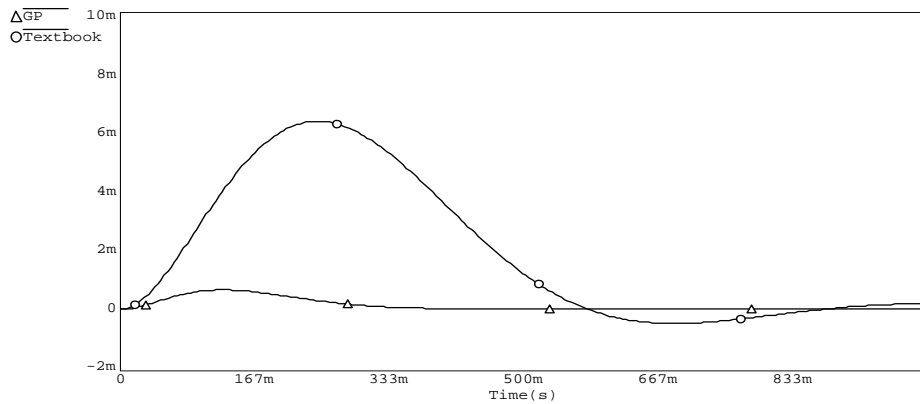
**COMPARISON OF THE TIME-DOMAIN  
RESPONSE TO 1-VOLT STEP INPUT FOR  
THE EVOLVED CONTROLLER  
(TRIANGLES) AND THE BISHOP AND  
DORF CONTROLLER (SQUARES) FOR  
THE TWO-LAG PLANT WITH  $K=1$  AND  
 $\tau=1$**



## OVERALL MODEL

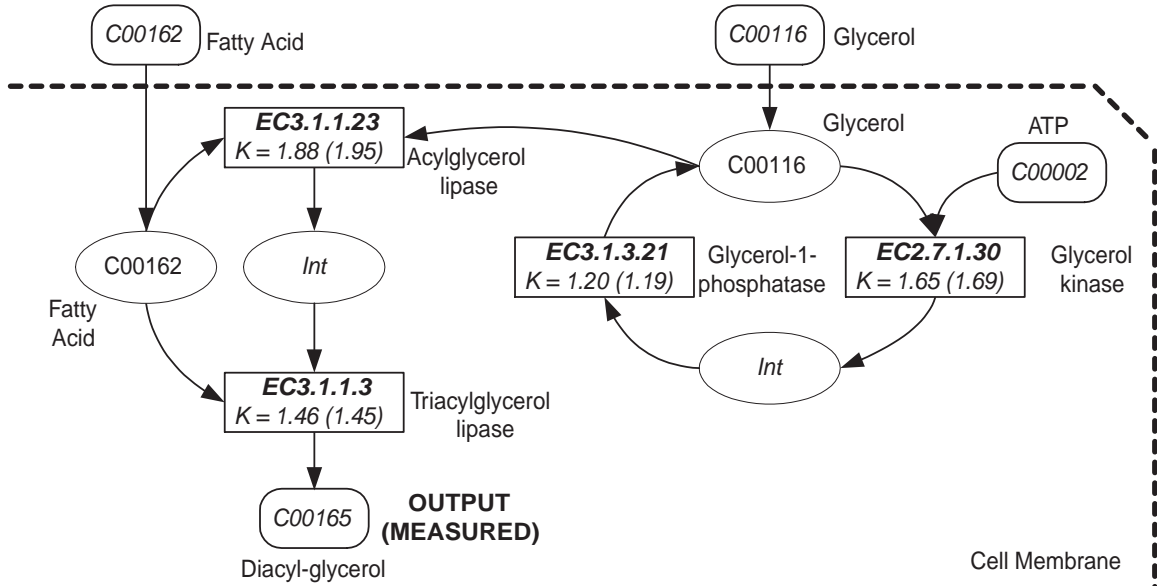


**COMPARISON OF THE TIME-DOMAIN  
RESPONSE TO A 1-VOLT DISTURBANCE  
SIGNAL OF THE EVOLVED  
CONTROLLER (TRIANGLES) AND THE  
BISHOP AND DORF CONTROLLER  
(CIRCLES) FOR THE TWO-LAG PLANT  
WITH  $K=1$  AND  $\tau=1$**

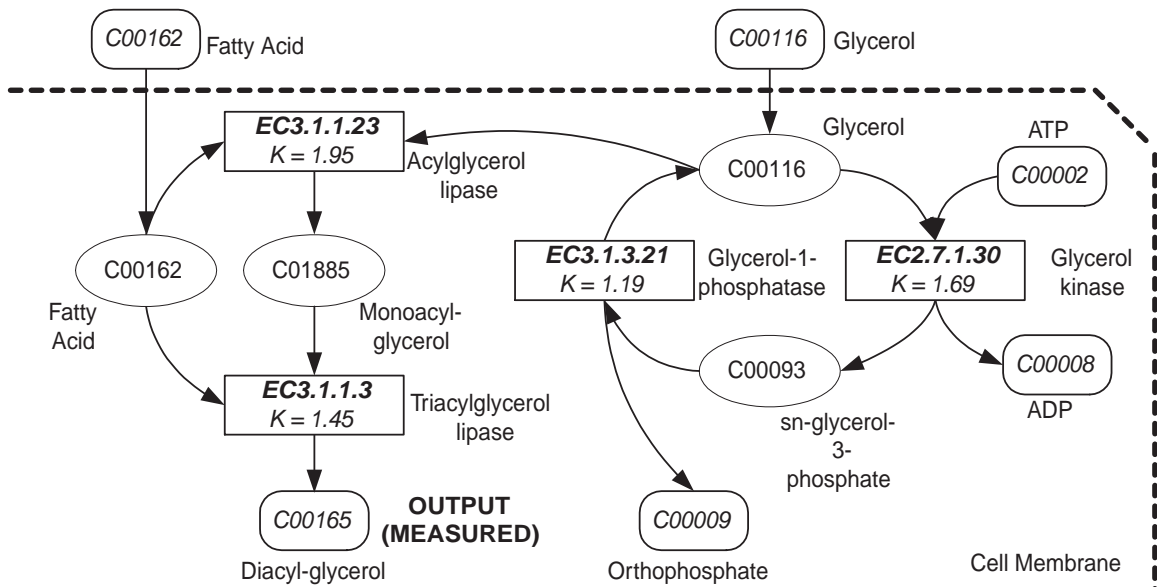


# REVERSE ENGINEERING OF METABOLIC PATHWAYS (4-REACTION NETWORK IN PHOSPHOLIPID CYCLE)

## BEST-OF-GENERATION 66



## DESIRED



## **CHARACTERISTICS SUGGESTING THE USE OF GENETIC PROGRAMMING**

- (1) discovering the size and shape of the solution,**
- (2) reusing substructures,**
- (3) discovering the number of substructures,**
- (4) discovering the nature of the hierarchical references among substructures,**
- (5) passing parameters to a substructure,**
- (6) discovering the type of substructures (e.g., subroutines, iterations, loops, recursions, or storage),**
- (7) discovering the number of arguments possessed by a substructure,**
- (8) maintaining syntactic validity and locality by means of a developmental process, or**
- (9) discovering a general solution in the form of a parameterized topology containing free variables**

## MANY DIFFERENT GA/ES ENCODINGS HAVE BEEN SUCCESSFULLY USED

**A mixture of real-valued variables, integer-valued variables, and categorical variables are encoded in the chromosome**

L	.220	2	3	C	403.	3	6	L	.528	6	9	L	.041	9	0
---	------	---	---	---	------	---	---	---	------	---	---	---	------	---	---

### • Bit-string chromosome

Resistor		2.5 $\Omega$								Node 3			Node 6		
0	1	0	0	1	0	1	0	0	0	0	1	1	1	1	0

- The component type (a categorical variable) is encoded as 2 bits (01 = resistor, etc.)
- The component value (real-valued number) is encoded as 8 bits
- The node (integer-valued variable) to which the component's 1<sup>st</sup> lead is connected is encoded by 3 bits
- The node (integer-valued variable) to which the component's 2<sup>nd</sup> lead is connected is encoded by 3 bits
- Note that the number of nodes is capped at 8 (or assumed to be 8)

# IT IS OFTEN POSSIBLE TO USE THE GENETIC ALGORITHM (GA) OR EVOLUTION STRATEGIES EVEN WHEN THE SIZE AND SHAPE OF THE SOLUTION IS A MAJOR ISSUE

- Variable-length genetic algorithm (VGA)
- Maintain constraints

### Chromosome #1

1 <sup>st</sup> Component				2 <sup>nd</sup> Component			
L	.220	1	2	C	403.	2	0

### Chromosome #2

1 <sup>st</sup> Component				2 <sup>nd</sup> Component			
R	250.	0	1	C	100.	1	2

### Nominal Offspring #1 is invalid

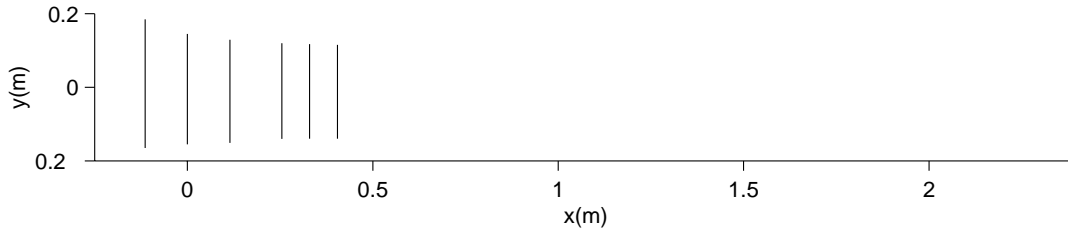
1 <sup>st</sup> Component				2 <sup>nd</sup> Component			
L	.220	1	2	C	100.	1	2

- Penalize (in fitness measure)
- Delete
- Repair (most common method)
- Inundate

## **STRONG INDICATIONS FOR USING GENETIC ALGORITHM (GA) OR EVOLUTION STRATEGIES (ES)**

- **The size and shape of the solution is known or fixed**
- **Ascertaining numerical parameters is the major issue**
- **Simplicity is a major consideration**
  - **On-chip evolution the algorithm's logic is implemented on the chip in hardware**

# AUTOMATIC SYNTHESIS OF A YAGI-UDA WIRE ANTENNA USING GENETIC ALGORITHM (LINDEN 1997)



- When the genetic algorithm (GA) operating on fixed-length character strings was used to synthesize a particular Yagi-Uda wire antenna by Linden (1997), the chromosome was based on
  - a particular number of reflectors (one) and
  - a particular number of directors.

The chromosome encoded

- the spacing between the parallel wires
- the length of each of the parallel wires



# **AUTOMATIC SYNTHESIS OF A YAGI-UDA WIRE ANTENNA USING GENETIC ALGORITHM (LINDEN 1997) — CONTINUED**

• When the genetic algorithm (GA) operating on fixed-length character strings was used to synthesize a Yagi-Uda wire antenna (Linden 1997), the following decisions were made by the human user prior to the start of the run:

- (1) the number of reflectors (one),
  - (2) the number of directors,
  - (3) the fact that the driven element, the directors, and the reflector are all single straight wires,
  - (4) the fact that the driven element, the directors, and the reflector are all arranged in parallel,
  - (5) the fact that the energy source (via the transmission line) is connected only to single straight wire (the driven element) — that is, all the directors and reflectors are parasitically coupled
- Characteristics (3), (4), and (5) are essential characteristics of the Yagi-Uda antenna, namely an antenna with multiple parallel parasitically coupled straight-line directors, a single parallel parasitically coupled straight-line reflector, and a straight-line driven element. That is, the GA run assumed that the answer would be a Yagi-Uda antenna.

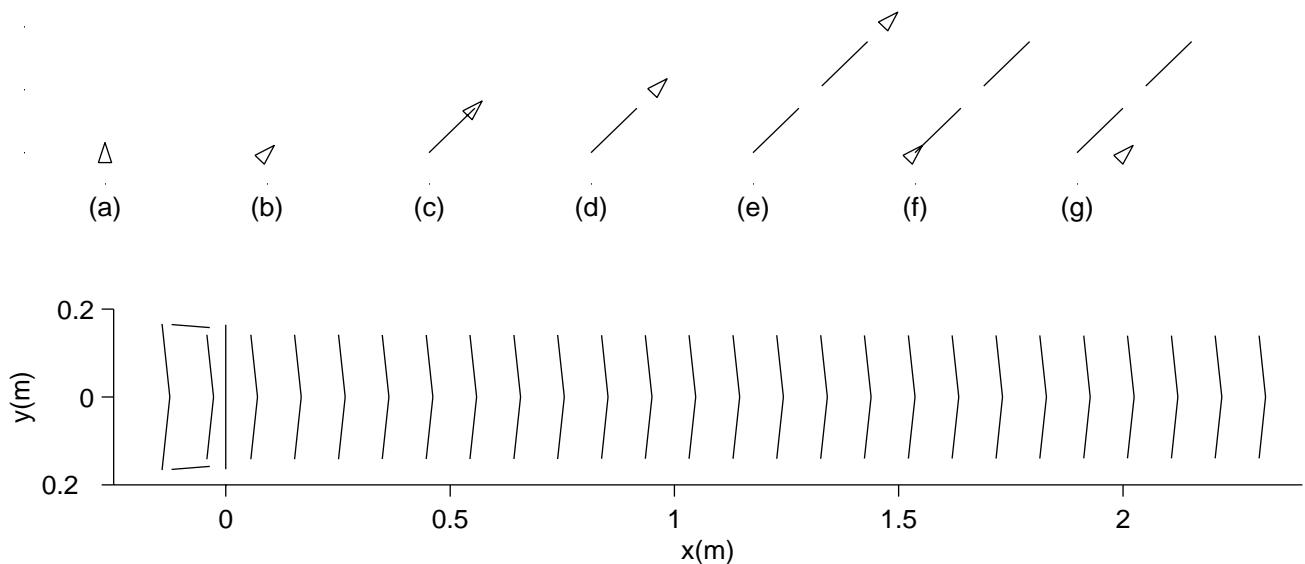
# AUTOMATIC SYNTHESIS OF A WIRE ANTENNA

## EXAMPLE OF TURTLE FUNCTIONS USED TO CREATE WIRE ANTENNA

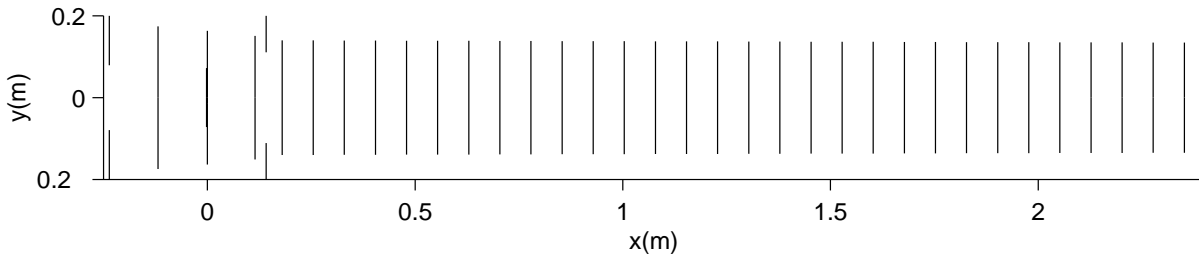
```

1 (PROGN3
2   (TURN-RIGHT 0.125)
3   (LANDMARK
4     (REPEAT 2
5       (PROGN2
6         (DRAW 1.0 HALF-MM-WIRE)
7         (DRAW 0.5 NO-WIRE)))
8   (TRANSLATE-RIGHT 0.125 0.75))

```



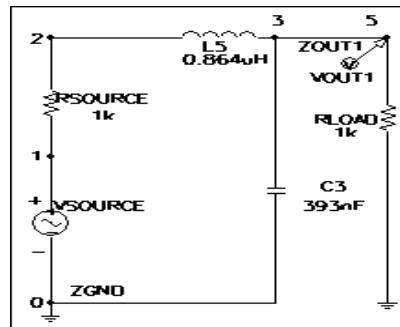
# BEST-OF-RUN ANTENNA FROM GENERATION 90 — FITNESS OF-16.04



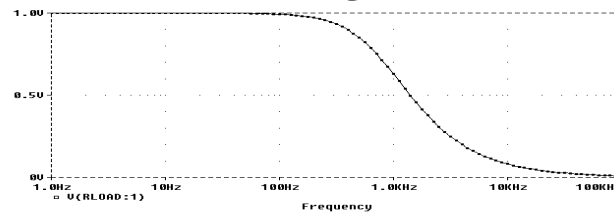
- **The GP run discovered**
  - (1) the number of reflectors (one),**
  - (2) the number of directors,**
  - (3) the fact that the driven element, the directors, and the reflector are all single straight wires,**
  - (4) the fact that the driven element, the directors, and the reflector are all arranged in parallel,**
  - (5) the fact that the energy source (via the transmission line) is connected only to single straight wire (the driven element) — that is, all the directors and reflectors are parasitically coupled**
  
- **Characteristics (3), (4), and (5) are essential characteristics of the Yagi-Uda antenna, namely an antenna with multiple parallel parasitically coupled straight-line directors, a single parallel parasitically coupled straight-line reflector, and a straight-line driven element.**

# REUSE LOWPASS FILTER USING ADFs

## GENERATION 0 – ONE-RUNG LADDER



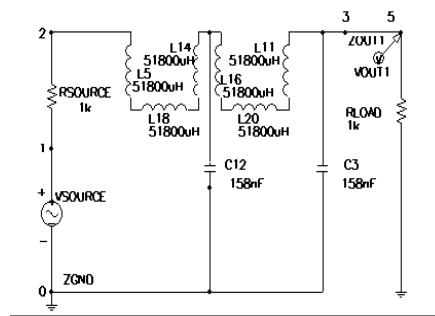
## BEHAVIOR IN FREQUENCY DOMAIN



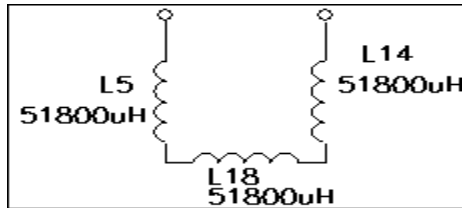
# REUSE

## LOWPASS FILTER USING ADFs

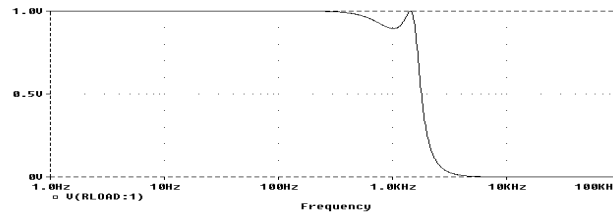
### GENERATION 9 - TWO-RUNG LADDER



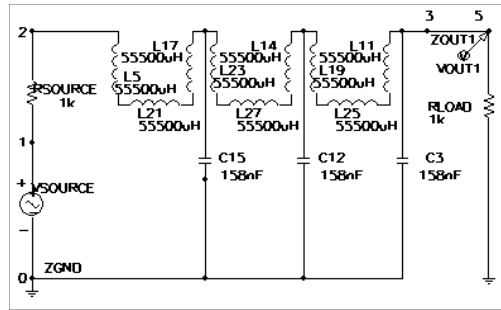
## TWICE-CALLED TWO-PORTED ADF0



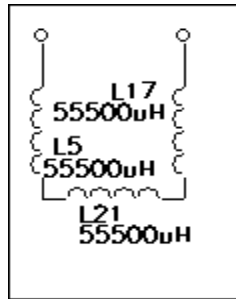
## BEHAVIOR IN FREQUENCY DOMAIN



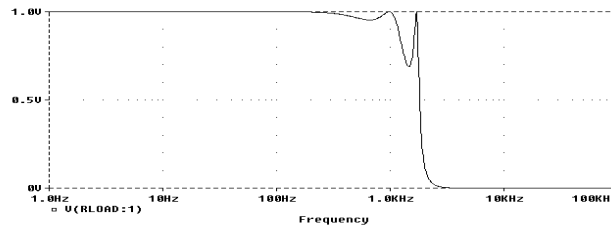
# REUSE LOWPASS FILTER USING ADFs GEN 16 – THREE-RUNG LADDER



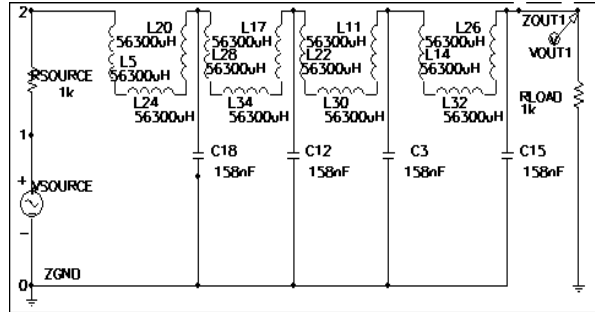
## THRICE-CALLED TWO-PORTED ADF<sup>0</sup>



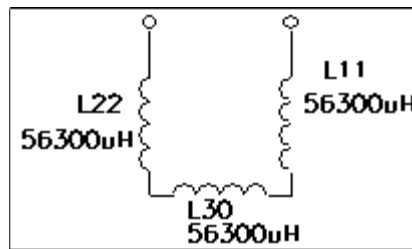
## BEHAVIOR IN FREQUENCY DOMAIN



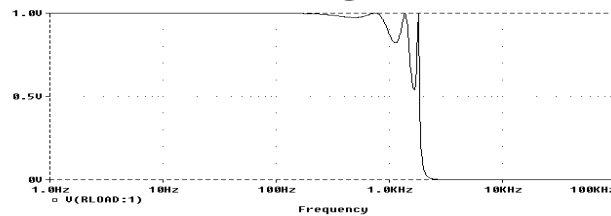
# REUSE LOWPASS FILTER USING ADFs GEN 20 – FOUR-RUNG LADDER



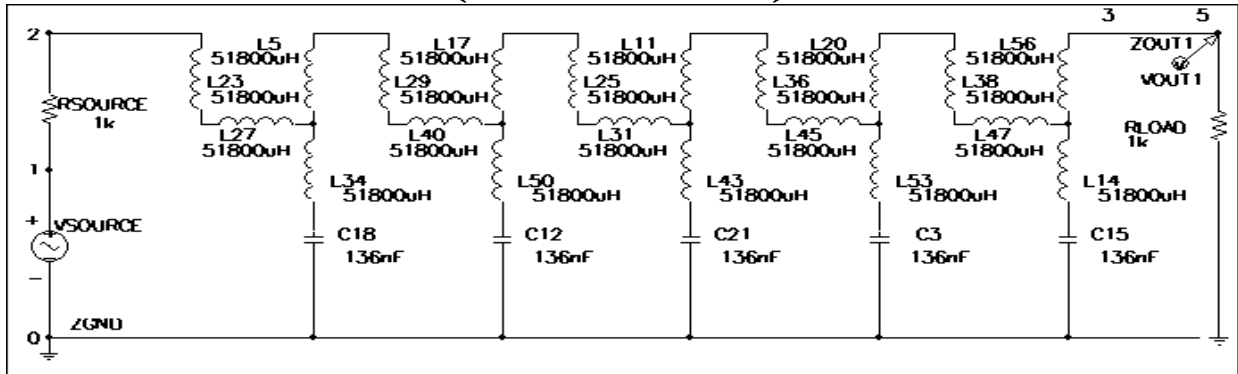
## QUADRUPLY-CALLED TWO-PORTED ADF<sup>0</sup>



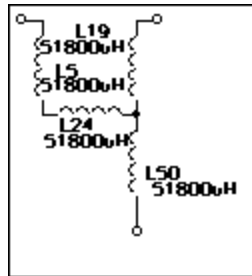
## BEHAVIOR IN FREQUENCY DOMAIN



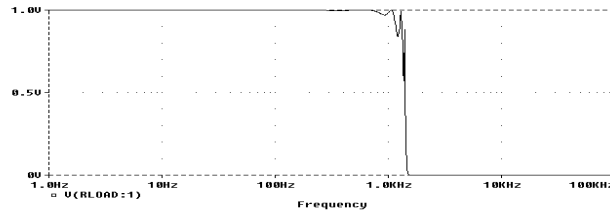
# REUSE LOWPASS FILTER USING ADFs GENERATION 31 — TOPOLOGY OF CAUER (ELLIPTIC) FILTER



## QUINTUPLY-CALLED THREE-PORTED ADF0



## BEHAVIOR IN FREQUENCY DOMAIN





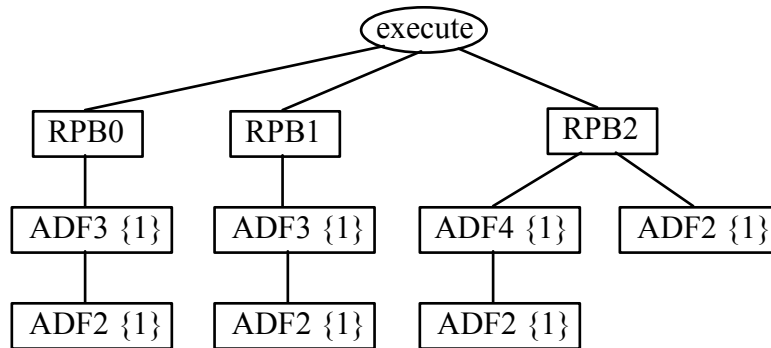
## PASSING A PARAMETER TO A SUBSTRUCTURE

- The set of potential terminals for each construction-continuing subtree of an automatically defined function,  $T_{\text{ccs-adf-potential}}$ , is

$$T_{\text{ccs-adf-potential}} = \{\mathbf{ARG0}\}$$

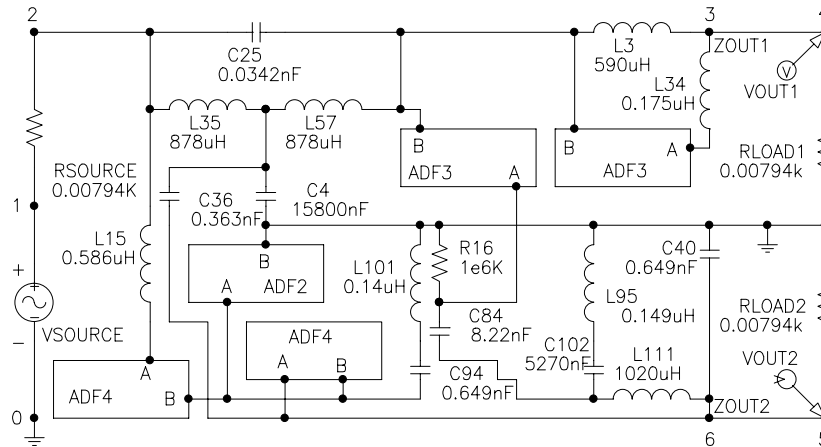
## EMERGENCE OF A PARAMETERIZED ARGUMENT IN A CIRCUIT SUBSTRUCTURE

## HIERARCHY OF BRANCHES FOR THE BEST-OF-RUN CIRCUIT- FROM GENERATION 158



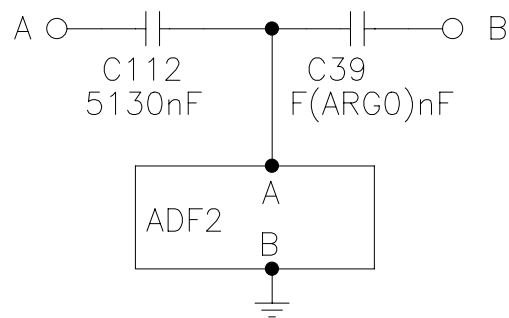
# PASSING A PARAMETER TO A SUBSTRUCTURE

## BEST-OF-RUN CIRCUIT FROM GENERATION 158



**THREE-PORTED AUTOMATICALLY  
DEFINED FUNCTION ADF3 OF THE  
BEST-OF-RUN CIRCUIT FROM  
GENERATION 158**

**ADF3 CONTAINS CAPACITOR C39  
PARAMETERIZED BY DUMMY  
VARIABLE ARG0**



## THE FIRST RESULT-PRODUCING BRANCH, RPB0, CALLING ADF3

```
(PARALLEL0 (L (+ (- 1.883196E-01 (- -9.095883E-02 5.724576E-01)) (- 9.737455E-01 -9.452780E-01)) (FLIP END)) (SERIES (C (+ (+ -6.668774E-01 -8.770285E-01) 4.587758E-02) (NOP END)) (SERIES END END (PARALLEL1 END END END END)) (FLIP (SAFE_CUT))) (PAIR_CONNECT_0 END END END) (PAIR_CONNECT_0 (L (+ -7.220122E-01 4.896697E-01) END) (L (- -7.195599E-01 3.651142E-02) (SERIES (C (+ -5.111248E-01 (- (- -6.137950E-01 -5.111248E-01) (- 1.883196E-01 (- -9.095883E-02 5.724576E-01)))) END) (SERIES END END (adf3 6.196514E-01)) (NOP END))) (NOP END)))
```

## AUTOMATICALLY DEFINED FUNCTION ADF3

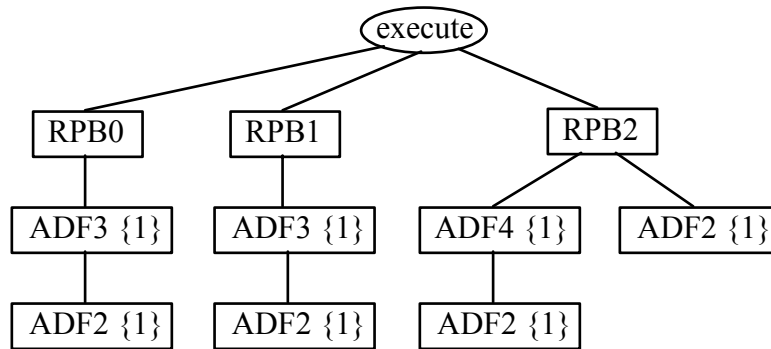
```
(C ((+ (- (+ (+ (+ 5.630820E-01 (- 9.737455E-01 -9.452780E-01)) (+ ARGO 6.953752E-02)) (- (- 5.627716E-02 (+ 2.273517E-01 (+ 1.883196E-01 (+ 9.346950E-02 (+ -7.220122E-01 (+ 2.710414E-02 1.397491E-02)))))) (- (+ (- 2.710414E-02 -2.807583E-01) (+ -6.137950E-01 -8.554120E-01)) (- -8.770285E-01 (- -4.049602E-01 -2.192044E-02)))))) (+ (+ 1.883196E-01 (+ (+ (+ (+ 9.346950E-02 (+ -7.220122E-01 (+ 2.710414E-02 1.397491E-02))) (- 4.587758E-02 -2.340137E-01)) 3.226026E-01) (+ -7.220122E-01 (- -9.131658E-01 6.595502E-01)))) 3.660116E-01)) 9.496355E-01) (THREE_GROUND_0 (C (+ (- (+ (+ (+ 5.630820E-01 (- 9.737455E-01 -9.452780E-01)) (+ (- (- -7.195599E-01 3.651142E-02) -9.761651E-01) (- (+ (- (- -7.195599E-01 3.651142E-02) -9.761651E-01) 6.953752E-02) 3.651142E-02))) (- (- 5.627716E-02 (- 1.883196E-01 (- -9.095883E-02 5.724576E-01))) (- (+ (- 2.710414E-02 -2.807583E-01) (+ -6.137950E-01 (+ ARGO 6.953752E-02))) (- -8.770285E-01 (- -4.049602E-01 -2.192044E-02)))))) (+ (+ 1.883196E-01 -7.195599E-01) 3.660116E-01)) 9.496355E-01) (NOP (FLIP (PAIR_CONNECT_0 END END END))) (FLIP (SERIES (FLIP (FLIP (FLIP END))) (C (- (+ 6.238477E-01 6.196514E-01) (+ (+ (- (- 4.037348E-01 4.343444E-01) (+ -7.788187E-01 (+ (+ (- -8.786904E-01 1.397491E-02) (- -6.137950E-01 (- (+ (- 2.710414E-02 -2.807583E-01) (+ -6.137950E-01 -8.554120E-01)) (- -8.770285E-01 (- -4.049602E-01 -2.192044E-02)))))) (+ (+ 7.215142E-03 1.883196E-01) (+ 7.733750E-01 4.343444E-01)))))) (- (- -9.389297E-01 5.630820E-01) (+ -5.840433E-02 3.568947E-01))) -8.554120E-01)) (NOP END)) END)) (FLIP (adf2 9.737455E-01)))
```

## **ADF3 DOES THREE THINGS**

- The structure that develops out of **ADF3** includes a capacitor **C112** whose value (5,130 uF) is not a function of its dummy variable, **ARG0**.
- The structure that develops out of **ADF3** has one hierarchical reference to **ADF2**. As previously mentioned, the invocation of **ADF2** is done with a constant (9.737455E-01) so this invocation of **ADF2** produces a 259  $\mu$ H inductor.
- Most importantly, the structure that develops out of **ADF3** creates a capacitor (**C39**) whose sizing, **F(ARG0)**, is a function of the dummy variable, **ARG0**, of automatically defined function **ADF3**. Capacitor **C39** has different sizing on different invocations of automatically defined function **ADF3**.
- The combined effect of **ADF3** is to insert the following three components:
  - an unparameterized 5,130 uF capacitor,
  - a parameterized capacitor **C39** whose component value is dependent on **ARG0** of **ADF3**, and
  - a parameterized inductor (created by **ADF2**) whose sizing is parameterized, but which, in practice, is called with a constant value.

# EMERGENCE OF A PARAMETERIZED ARGUMENT IN A CIRCUIT SUBSTRUCTURE

## HIERARCHY OF BRANCHES FOR THE BEST-OF-RUN CIRCUIT- FROM GENERATION 158



## FREE VARIABLE (INPUT) AND CONDITIONALS

### SOLVING A QUADRATIC EQUATION USING THE GENETIC ALGORITHM

- Suppose we want the 2 roots of the quadratic equation

$$1x^2 - 3x + 2 = 0$$

- Using the genetic algorithm (GA) operating on a fixed-length character string, we can search a space of encodings using an alphabet size of 2 (i.e., binary) of length, say, 16 representing two real numbers (each with, say, 4 bits to left of the "decimal" point). After running the GA, a solution is

0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0					
								↓									↓				
								1.0									2.0				

- Alternatively, we could use a "floating point" genetic algorithm (GA) to search a space of 2-part encodings. A solution is

1.0	2.0
-----	-----

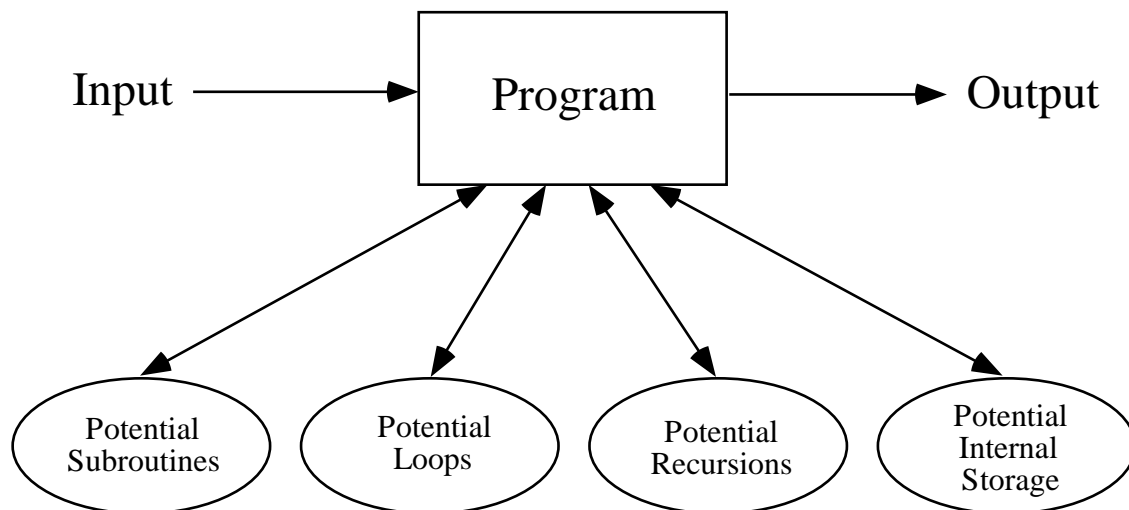
- In either case, the result is a solution to ONE INSTANCE of the quadratic equation problem.

# SOLVING A QUADRATIC EQUATION USING GENETIC PROGRAMMING (GP)

- Using genetic programming (GP), we can solve the general, parameterized quadratic equation

$$ax^2 + bx + c = 0$$

by searching the space of computer programs for a program that takes  $a$ ,  $b$ , and  $c$  as inputs

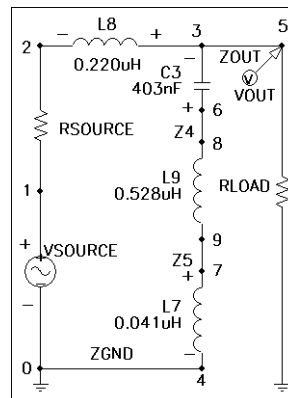


- The result is a solution to ALL INSTANCES of the quadratic equation problem



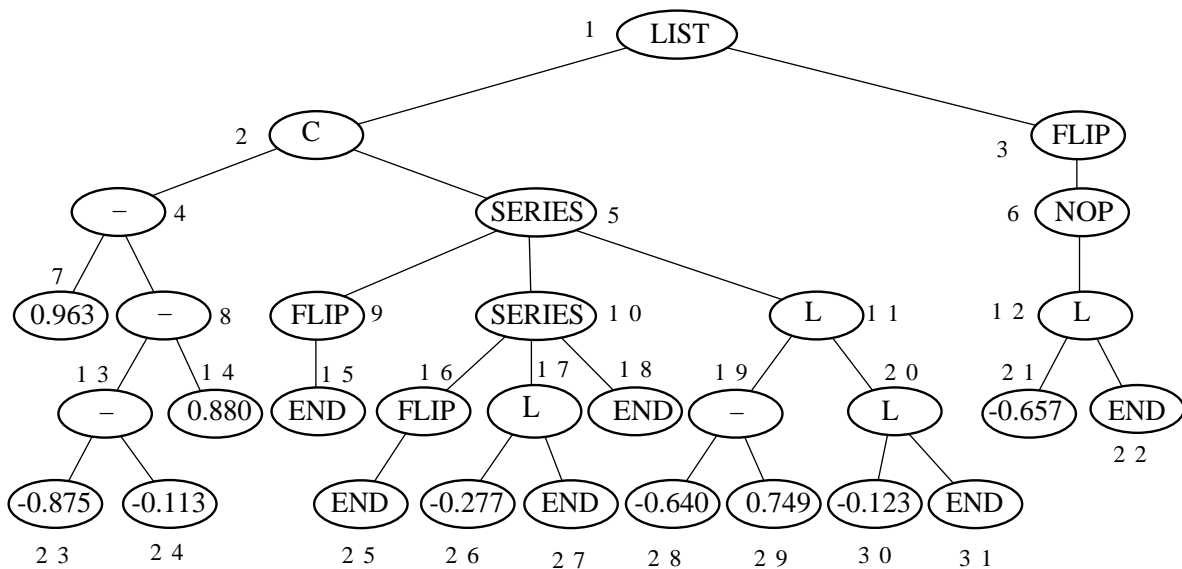
**GENERAL APPEARANCE OF ONE  
POSSIBLE CHROMOSOME ENCODING  
USED TO SOLVE ONE INSTANCE OF A  
CIRCUIT PROBLEM USING THE  
GENETIC ALGORITHM (GA)  
OPERATING ON FIXED-LENGTH  
CHARACTER STRINGS**

**EXAMPLE CIRCUIT**



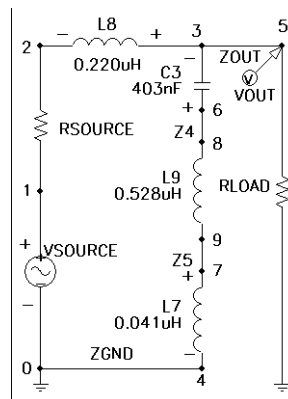
1 <sup>st</sup> Component				2 <sup>nd</sup> Component				3 <sup>rd</sup> Component				4 <sup>th</sup> Component			
L	.220	2	3	C	403.	3	6	L	.528	6	9	L	.041	9	0

# THE GENERAL APPEARANCE OF EXPRESSIONS USED TO SOLVE ONE INSTANCE OF A CIRCUIT PROBLEM USING GENETIC PROGRAMMING (GP) IN *GENETIC PROGRAMMING III* (1999)



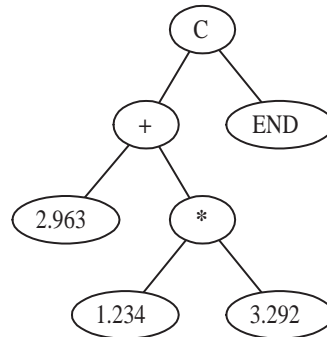
```
(LIST (C (- 0.963 (- (- -0.875 -0.113) 0.880)) (series (flip
end) (series (flip end) (L -0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip (nop (L -0.657 end))))))
```

## EXAMPLE CIRCUIT (GEN 0)

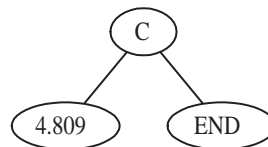


# VALUE-SETTING SUBTREES—3 WAYS

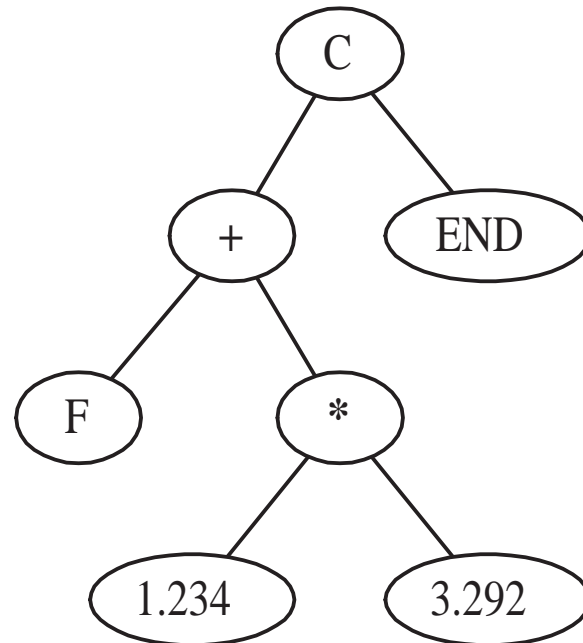
## ARITHMETIC-PERFORMING SUBTREE



## SINGLE PERTURBABLE CONSTANT



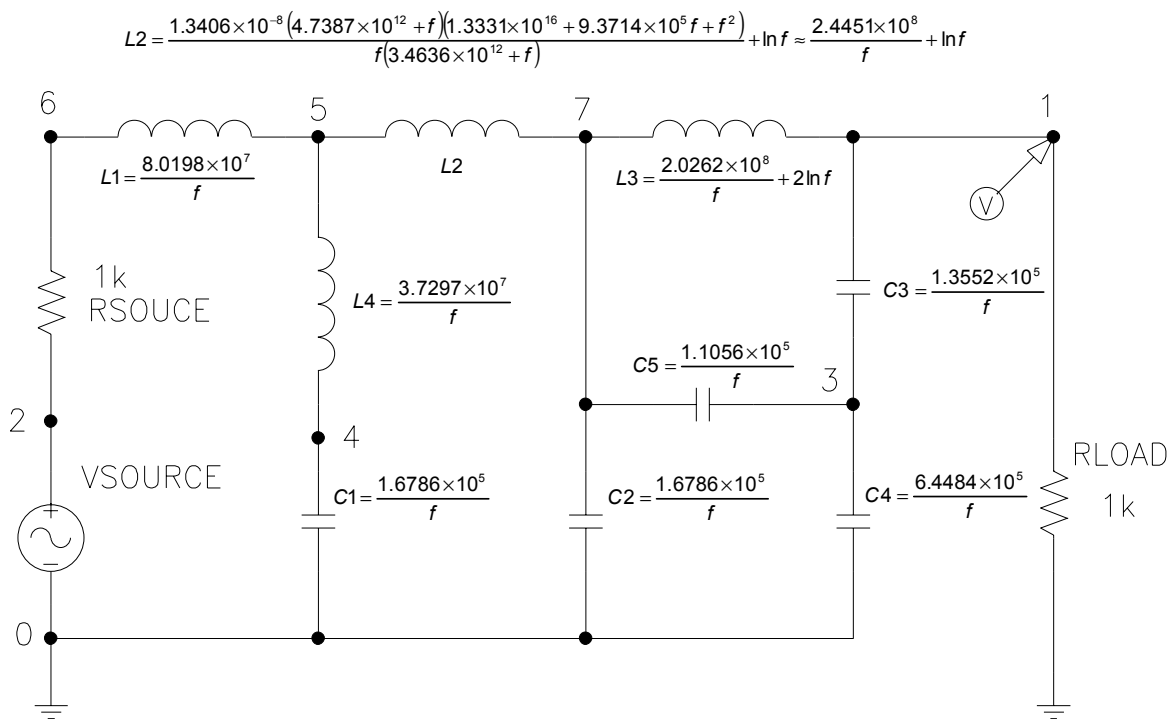
## FREE VARIABLE



# PARAMETERIZED TOPOLOGY FOR "GENERALIZED" LOWPASS FILTER

## VARIABLE CUTOFF LOWPASS FILTER

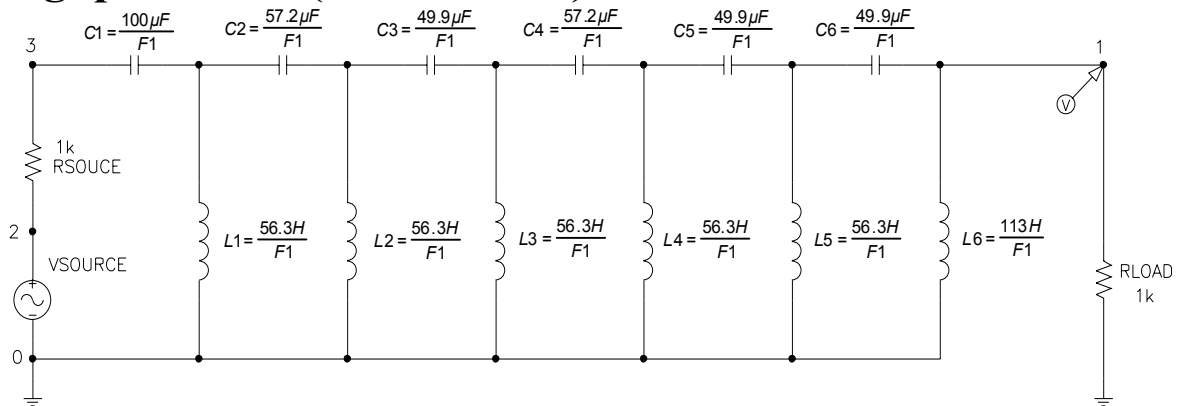
- Want lowpass filter whose passband ends at frequencies  $f = 1,000, 1,780, 3,160, 5,620, 10,000, 17,800, 31,600, 56,200, 100,000$  Hz



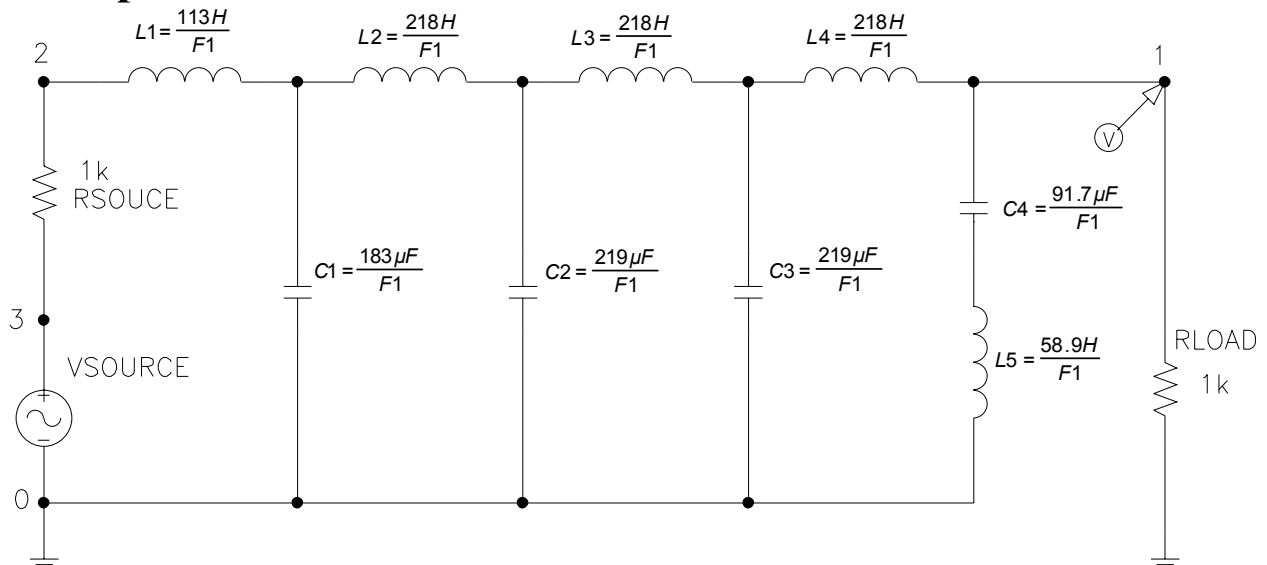
# PARAMETERIZED TOPOLOGY USING CONDITIONAL DEVELOPMENTAL OPERATORS (GENETIC SWITCH)

## VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT

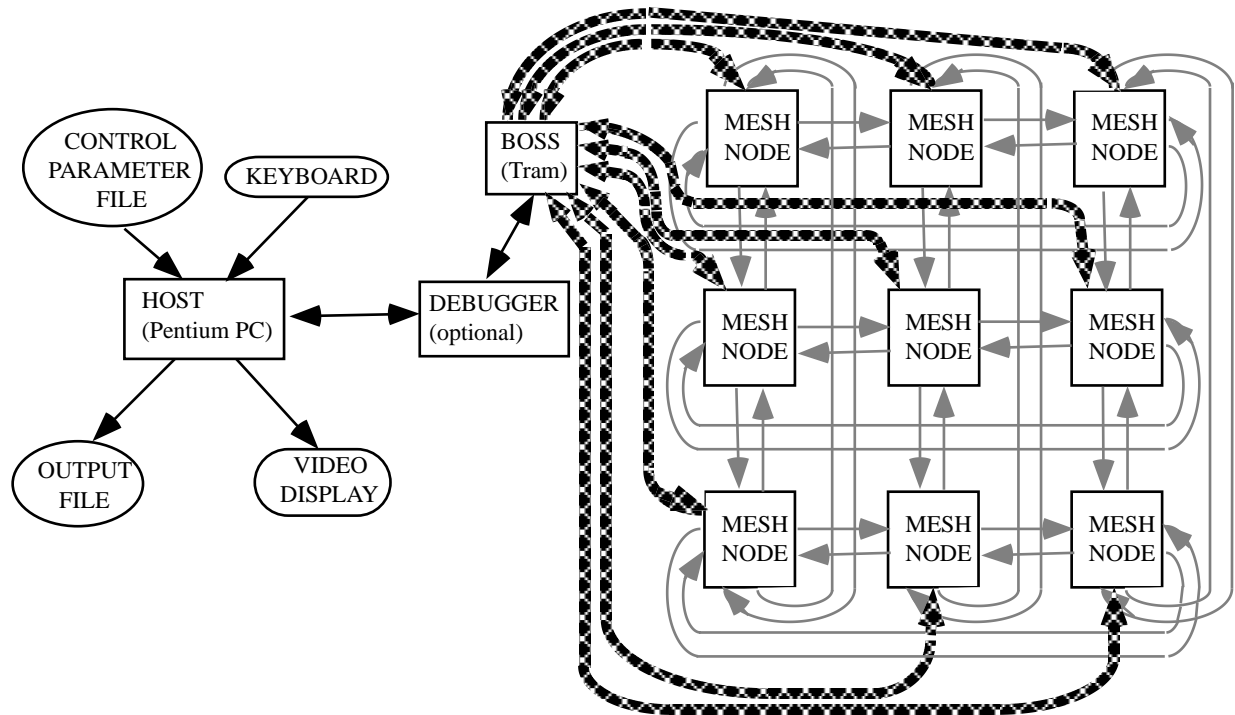
- Best-of-run circuit from generation 93 when inputs call for a highpass filter (i.e.,  $F1 > F2$ ).



- Best-of-run circuit from generation 93 when inputs call for a lowpass filter.



# PARALLELIZATION BY SUBPOPULATIONS ("ISLAND" OR "DEME" MODEL OR "DISTRIBUTED GENETIC ALGORITHM")



- Like Hormel, Get Everything Out of the Pig, Including the Oink
- Keep on Trucking
- It Takes a Licking and Keeps on Ticking
- The Whole is Greater than the Sum of the Parts

## PETA-OPS

- Human brain operates at  $10^{12}$  neurons operating at  $10^3$  per second =  $10^{15}$  ops per second
- $10^{15}$  ops = 1 peta-op = 1 bs (brain second)

## GENETIC PROGRAMMING OVER 15- YEAR PERIOD 1987–2002

<b>System</b>	<b>Period of usage</b>	<b>Petacycles (<math>10^{15}</math> cycles) per day for entire system</b>	<b>Speed-up over previous system</b>	<b>Speed-up over first system in this table</b>	<b>Human-competitive results</b>
<b>Serial Texas Instruments LISP machine</b>	<b>1987–1994</b>	<b>0.00216</b>	<b>1 (base)</b>	<b>1 (base)</b>	<b>0</b>
<b>64-node Transtech transputer parallel machine</b>	<b>1994–1997</b>	<b>0.02</b>	<b>9</b>	<b>9</b>	<b>2</b>
<b>64-node Parsytec parallel machine</b>	<b>1995–2000</b>	<b>0.44</b>	<b>22</b>	<b>204</b>	<b>12</b>
<b>70-node Alpha parallel machine</b>	<b>1999–2001</b>	<b>3.2</b>	<b>7.3</b>	<b>1,481</b>	<b>2</b>
<b>1,000-node Pentium II parallel machine</b>	<b>2000–2002</b>	<b>30.0</b>	<b>9.4</b>	<b>13,900</b>	<b>12</b>

## PROGRESSION OF RESULTS

System	Period	Speed-up	Qualitative nature of the results produced by genetic programming
Serial LISP machine	1987–1994	1 (base)	<ul style="list-style-type: none"> <li>• Toy problems of the 1980s and early 1990s from the fields of artificial intelligence and machine learning</li> </ul>
64-node Transtech 8-biy transputer	1994–1997	9	<ul style="list-style-type: none"> <li>• Two human-competitive results involving one-dimensional discrete data (not patent-related)</li> </ul>
64-node Parsytec parallel machine	1995–2000	22	<ul style="list-style-type: none"> <li>• One human-competitive result involving two-dimensional discrete data</li> <li>• Numerous human-competitive results involving continuous signals analyzed in the frequency domain</li> <li>• Numerous human-competitive results involving 20<sup>th</sup>-century patented inventions</li> </ul>
70-node Alpha parallel machine	1999–2001	7.3	<ul style="list-style-type: none"> <li>• One human-competitive result involving continuous signals analyzed in the time domain</li> <li>• Circuit synthesis extended from topology and sizing to include routing and placement (layout)</li> </ul>
1,000-node Pentium II parallel machine	2000–2002	9.4	<ul style="list-style-type: none"> <li>• Numerous human-competitive results involving continuous signals analyzed in the time domain</li> <li>• Numerous general solutions to problems in the form of parameterized topologies</li> <li>• Six human-competitive results duplicating the functionality of 21<sup>st</sup>-century patented inventions</li> </ul>
Long (4-week) runs of 1,000-node Pentium II parallel machine	2002	9.3	<ul style="list-style-type: none"> <li>• Generation of two patentable new inventions</li> </ul>



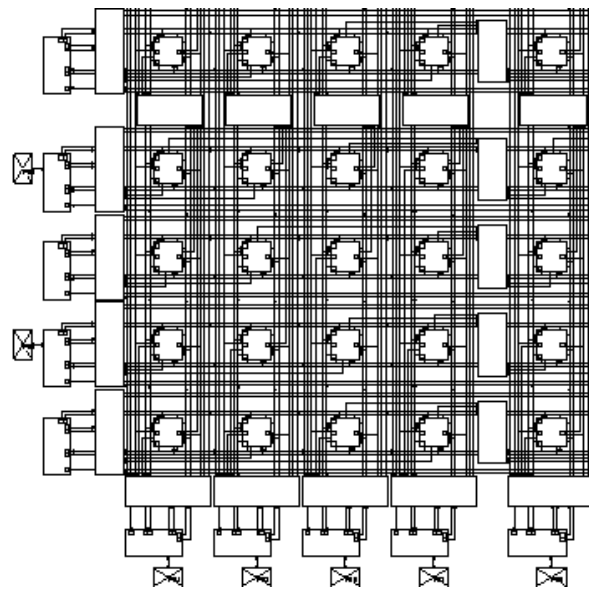
**PROGRESSION OF QUALITATIVELY  
MORE SUBSTANTIAL RESULTS  
PRODUCED BY GENETIC  
PROGRAMMING IN RELATION TO FIVE  
ORDER-OF-MAGNITUDE INCREASES IN  
COMPUTATIONAL POWER**

- **toy problems**
- **human-competitive results not related to patented inventions**
- **20<sup>th</sup>-century patented inventions**
- **21<sup>st</sup>-century patented inventions**
- **patentable new inventions**

# EVOLVABLE HARDWARE

## RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

### SMALL 5 BY 5 CORNER OF XILINX XC6216 FPGA



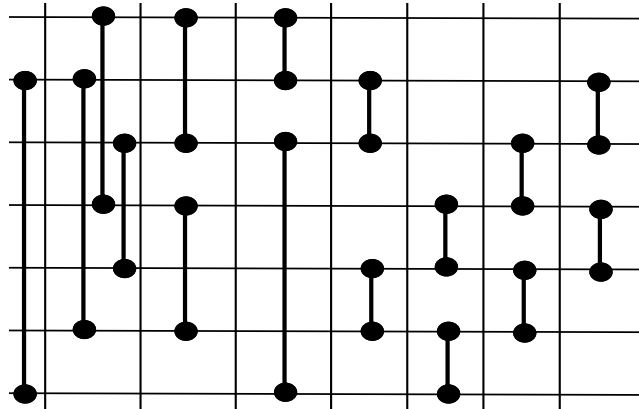
# EVOLVABLE HARDWARE

## RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

### SORTING NETWORKS

- A 16-step 7-sorter was evolved that has two fewer steps than the sorting network described in O'Connor and Nelsons' patent (1962) and that has the same number of steps as the 7-sorter that was devised by Floyd and Knuth subsequent to the patent and described in Knuth 1973.

### GENETICALLY EVOLVED 7-SORTER



## **FUNDAMENTAL DIFFERENCES BETWEEN GP AND OTHER APPROACHES TO AI AND ML**

- (1) Representation: Genetic programming overtly conducts its search for a solution to the given problem in program space.**
- (2) Role of point-to-point transformations in the search: Genetic programming does not conduct its search by transforming a single point in the search space into another single point, but instead transforms a set of points into another set of points.**
- (3) Role of hill climbing in the search: Genetic programming does not rely exclusively on greedy hill climbing to conduct its search, but instead allocates a certain number of trials, in a principled way, to choices that are known to be inferior.**
- (4) Role of determinism in the search: Genetic programming conducts its search probabilistically.**
- (5) Role of an explicit knowledge base: None.**
- (6) Role of formal logic in the search: None.**
- (7) Underpinnings of the technique: Biologically inspired.**

## 36 HUMAN-COMPETITIVE RESULTS (LIST AS OF MAY 2003)

	Claimed instance	Basis for claim of human-competitiveness	Reference
1	Creation of a better-than-classical quantum algorithm for the Deutsch-Jozsa “early promise” problem	B, F	Spector, Barnum, and Bernstein 1998
2	Creation of a better-than-classical quantum algorithm for Grover’s database search problem	B, F	Spector, Barnum, and Bernstein 1999
3	Creation of a quantum algorithm for the depth-two AND/OR query problem that is better than any previously published result	D	Spector, Barnum, Bernstein, and Swamy 1999; Barnum, Bernstein, and Spector 2000
4	Creation of a quantum algorithm for the depth-one OR query problem that is better than any previously published result	D	Barnum, Bernstein, and Spector 2000
5	Creation of a protocol for communicating information through a quantum gate that was previously thought not to permit such communication	D	Spector and Bernstein 2003
6	Creation of a novel variant of quantum dense coding	D	Spector and Bernstein 2003
7	Creation of a soccer-playing program that won its first two games in the Robo Cup 1997 competition	H	Luke 1998
8	Creation of a soccer-playing program that ranked in the middle of the field of 34 human-written programs in the Robo Cup 1998 competition	H	Andre and Teller 1999
9	Creation of four different algorithms for the transmembrane segment identification problem for proteins	B, E	Sections 18.8 and 18.10 of <i>GP-2 book</i> and sections 16.5 and 17.2 of <i>GP-3 book</i>
10	Creation of a sorting network for seven items using only 16 steps	A, D	Sections 21.4.4, 23.6, and 57.8.1 of <i>GP-3 book</i>
11	Rediscovery of the Campbell ladder topology for lowpass and highpass filters	A, F	Section 25.15.1 of <i>GP-3 book</i> and section 5.2 of <i>GP-4 book</i>
12	Rediscovery of the Zobel “ <i>M</i> -derived half section” and “constant <i>K</i> ” filter sections	A, F	Section 25.15.2 of <i>GP-3 book</i>
13	Rediscovery of the Cauer (elliptic) topology for filters	A, F	Section 27.3.7 of <i>GP-3 book</i>
14	Automatic decomposition of the problem of synthesizing a crossover filter	A, F	Section 32.3 of <i>GP-3 book</i>
15	Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of an amplifier and other circuits	A, F	Section 42.3 of <i>GP-3 book</i>
16	Synthesis of 60 and 96 decibel amplifiers	A, F	Section 45.3 of <i>GP-3 book</i>
17	Synthesis of analog computational circuits for squaring, cubing, square root, cube root, logarithm, and Gaussian functions	A, D, G	Section 47.5.3 of <i>GP-3 book</i>
18	Synthesis of a real-time analog circuit for time-optimal control of a robot	G	Section 48.3 of <i>GP-3 book</i>

19	Synthesis of an electronic thermometer	A, G	Section 49.3 of GP-3 book
20	Synthesis of a voltage reference circuit	A, G	Section 50.3 of GP-3 book
21	Creation of a cellular automata rule for the majority classification problem that is better than the Gacs-Kurdyumov-Levin (GKL) rule and all other known rules written by humans	D, E	Andre, Bennett, and Koza 1996 and section 58.4 of GP-3 book
22	Creation of motifs that detect the D-E-A-D box family of proteins and the manganese superoxide dismutase family	C	Section 59.8 of GP-3 book
23	Synthesis of topology for a PID-D2 (proportional, integrative, derivative, and second derivative) controller	A, F	Section 3.7 of GP-4 book
24	Synthesis of an analog circuit equivalent to Philbrick circuit	A, F	Section 4.3 of GP-4 book
25	Synthesis of a NAND circuit	A, F	Section 4.4 of GP-4 book
26	Simultaneous synthesis of topology, sizing, placement, and routing of analog electrical circuits	A, F, G	Chapter 5 of GP-4 book
27	Synthesis of topology for a PID (proportional, integrative, and derivative) controller	A, F	Section 9.2 of GP-4 book
28	Rediscovery of negative feedback	A, E, F, G	Chapter 14 of GP-4 book
29	Synthesis of a low-voltage balun circuit	A	Section 15.4.1 of GP-4 book
30	Synthesis of a mixed analog-digital variable capacitor circuit	A	Section 15.4.2 of GP-4 book
31	Synthesis of a high-current load circuit	A	Section 15.4.3 of GP-4 book
32	Synthesis of a voltage-current conversion circuit	A	Section 15.4.4 of GP-4 book
33	Synthesis of a Cubic function generator	A	Section 15.4.5 of GP-4 book
34	Synthesis of a tunable integrated active filter	A	Section 15.4.6 of GP-4 book
35	Creation of PID tuning rules that outperform the Ziegler-Nichols and Åström-Hägglund tuning rules	A, B, D, E, F, G	Chapter 12 of GP-4 book
36	Creation of three non-PID controllers that outperform a PID controller that uses the Ziegler-Nichols or Åström-Hägglund tuning rules	A, B, D, E, F, G	Chapter 13 of GP-4 book

## **PROMISING GP APPLICATION AREAS**

- **Problem areas involving many variables that are interrelated in highly non-linear ways**
- **Inter-relationship of variables is not well understood**
- **A good approximate solution is satisfactory**
  - **design**
  - **control**
  - **classification and pattern recognition**
  - **data mining**
  - **system identification and forecasting**
- **Discovery of the size and shape of the solution is a major part of the problem**
- **Areas where humans find it difficult to write programs**
  - **parallel computers**
  - **cellular automata**
  - **multi-agent strategies / distributed AI**
  - **FPGAs**
- **"black art" problems**
  - **synthesis of topology and sizing of analog circuits**
  - **synthesis of topology and tuning of controllers**
  - **quantum computing circuits**
  - **synthesis of designs for antennas**
- **Areas where you simply have no idea how to program a solution, but where the objective (fitness measure) is clear**
- **Problem areas where large computerized databases are accumulating and computerized techniques are needed to analyze the data**

## TURING'S THREE APPROACHES TO MACHINE INTELLIGENCE

- Turing made the connection between searches and the challenge of getting a computer to solve a problem without explicitly programming it in his 1948 essay "Intelligent Machines" (in *Mechanical Intelligence: Collected Works of A. M. Turing*, 1992, edited by D. C. Ince).

"Further research into intelligence of machinery will probably be very greatly concerned with 'searches' ..."

## TURING'S THREE APPROACHES TO MACHINE INTELLIGENCE — CONTINUED

### 1. LOGIC-BASED SEARCH

One approach that Turing identified is a search through the space of integers representing candidate computer programs.

### 2. CULTURAL SEARCH

Another approach is the "cultural search" which relies on knowledge and expertise acquired over a period of years from others (akin to present-day knowledge-based systems).



# **TURING'S THREE APPROACHES TO MACHINE INTELLIGENCE — CONTINUED**

## **3. GENETICAL OR EVOLUTIONARY SEARCH**

**"There is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being the survival value."**

- **from Turing's 1950 paper "Computing Machinery and Intelligence" ...**

**"We cannot expect to find a good child-machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution, by the identifications"**

**"Structure of the child machine = Hereditary material"**

**"Changes of the child machine = Mutations"**

**"Natural selection = Judgment of the experimenter"**

## 16 AUTHORED BOOKS ON GP

- Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. 1998. *Genetic Programming - An Introduction*. San Francisco, CA: [Morgan Kaufman Publishers](#) and Heidelberg, Germany: dpunkt.verlag.
- Babovic, Vladan. 1996b. *Emergence, Evolution, Intelligence: Hydroinformatics*. Rotterdam, The Netherlands: Balkema Publishers.
- Blickle, Tobias. 1997. *Theory of Evolutionary Algorithms and Application to System Synthesis*. TIK-Schriftenreihe Nr. 17. Zurich, Switzerland: [vdf Hochschul Verlag AG and der ETH Zurich](#). ISBN 3-7281-2433-8.
- Jacob, Christian. 1997. *Principia Evolvica: Simulierte Evolution mit Mathematica*. Heidelberg, Germany: dpunkt.verlag. In German. English translation forthcoming in 2000 from [Morgan Kaufman Publishers](#).
- Jacob, Christian. 2001. *Illustrating Evolutionary Computation with Mathematica*. San Francisco: Morgan Kaufmann.
- Iba, Hitoshi. 1996. *Genetic Programming*. Tokyo: Tokyo Denki University Press. In Japanese.
- Koza, John R. 1992. [Genetic Programming: On the Programming of Computers by Means of Natural Selection](#). Cambridge, MA: The MIT Press.
- Koza, John R. 1994a. [Genetic Programming II: Automatic Discovery of Reusable Programs](#). Cambridge, MA: The MIT Press
- Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1999a. [Genetic Programming III: Darwinian Invention and Problem Solving](#). San Francisco, CA: Morgan Kaufmann Publishers.
- Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido. 2003. *Genetic Programming IV. Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.
- Langdon, William B. 1998. [Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!](#) Amsterdam: Kluwer Academic Publishers.
- Langdon, William B. and Poli, Riccardo. 2002. *Foundations of Genetic Programming*. Berlin: Springer-Verlag.
- Nordin, Peter. 1997. *Evolutionary Program Induction of Binary Machine Code and its Application*. Munster, Germany: Krehl Verlag.
- O'Neill, Michael and Ryan, Conor. 2003. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Boston: Kluwer Academic Publishers.
- Ryan, Conor. 1999. *Automatic Re-engineering of Software Using Genetic Programming*. Amsterdam: Kluwer Academic Publishers.
- Wong, Man Leung and Leung, Kwong Sak. 2000. *Data Mining Using Grammar Based Genetic Programming and Applications*. Amsterdam: Kluwer Academic Publishers.

# CONFERENCE PROCEEDINGS

- Banzhaf, Wolfgang, Daida, Jason, Eiben, A. E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E. (editors). 1999. *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, July 13-17, 1999, Orlando, Florida USA*. San Francisco, CA: Morgan Kaufmann.
- Banzhaf, Wolfgang, Poli, Riccardo, Schoenauer, Marc, and Fogarty, Terence C. 1998. *Genetic Programming: First European Workshop. EuroGP'98. Paris, France*. Lecture Notes in Computer Science. Volume 1391. Berlin, Germany: Springer-Verlag.
- ~~Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors).~~  
1996. *Genetic Programming 1996: Proceedings of the First Annual Conference*. Cambridge, MA: The MIT Press.
- Foster, James A., Lutton, Evelyne, Miller, Julian, Ryan, Conor, and Tettamanzi, Andrea G. B. (editors). 2002. *Genetic Programming: 5th European Conference, EuroGP 2002, Kinsale, Ireland, April 2002 Proceedings*. Berlin: Springer.
- Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (editors). 1997. *Genetic Programming 1997: Proceedings of the Second Annual Conference*. San Francisco, CA: Morgan Kaufmann.
- Koza, John R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, and Riolo, Rick. (editors). 1998. *Genetic Programming 1998: Proceedings of the Third Annual Conference*. San Francisco, CA: Morgan Kaufmann.
- Miller, Julian, Tomassini, Marco, Lanzi, Pier Luca, Ryan, Conor, Tettamanzi, Andrea G. B., and Langdon, William B. (editors). 2001. *Genetic Programming: 4<sup>th</sup> European Conference, EuroGP 2001, Lake Como, Italy, April 2001 Proceedings*. Berlin: Springer.
- Poli, Riccardo, Nordin, Peter, Langdon, William B., and Fogarty, Terence C. 1999. *Genetic Programming: Second European Workshop. EuroGP'99. Goteborg, Sweden, May 1999*. Lecture Notes in Computer Science. Volume 1598. Berlin, Germany: Springer-Verlag.
- Poli, Riccardo, Banzhaf, Wolfgang, Langdon, William B., Miller, Julian, Nordin, Peter, and Fogarty, Terence C. 2000. *Genetic Programming: European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 2000, Proceedings*. Lecture Notes in Computer Science. Volume 1802. Berlin, Germany: Springer-Verlag. ISBN 3-540-67339-3.
- Riolo, Rich and Worzel, William. 2003. *Genetic Programming: Theory and Practice*. Boston: Kluwer Academic Publishers.
- Spector, Lee, Goodman, E., Wu, A., Langdon, William B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, Marco, Pezeshk, S., Garzon, Max, and Burke, E. (editors). 2001. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*. San Francisco, CA: Morgan Kaufmann Publishers. Pages 57 - 65. Whitley, Darrell, Goldberg, David, Cantu-Paz, Erick, Spector, Lee, Parmee, Ian, and Beyer, Hans-Georg (editors). *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, July 10 - 12, 2000, Las Vegas, Nevada*. San Francisco: Morgan Kaufmann Publishers.

### **3 EDITED *ADVANCES IN GENETIC PROGRAMMING* BOOKS**

Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge, MA: The MIT Press.

Kinnear, Kenneth E. Jr. (editor). 1994. *Advances in Genetic Programming*. Cambridge, MA: The MIT Press.

Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter (editors). 1999. *Advances in Genetic Programming 3*. Cambridge, MA: The MIT Press.

### **4 VIDEOTAPES ON GP**

Koza, John R., and Rice, James P. 1992. [\*Genetic Programming: The Movie\*](#). Cambridge, MA: The MIT Press.

Koza, John R. 1994b. [\*Genetic Programming II Videotape: The Next Generation\*](#). Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, Keane, Martin A., and Brave, Scott. 1999. *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. San Francisco, CA: Morgan Kaufmann Publishers.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, Lanza, Guido, and Fletcher, David. 2003. *Genetic Programming IV Video: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

# **WILLIAM LANGDON'S BIBLIOGRAPHY ON GENETIC PROGRAMMING**

This bibliography is the most extensive in the field and contains over 3,034 papers (as of January 2003) by over 880 authors.

Visit

<http://www.cs.bham.ac.uk/~wbl/biblio/>

or

<http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html>

## ***GENETIC PROGRAMMING AND EVOLVABLE MACHINES* JOURNAL FROM KLUWER ACADEMIC PUBLISHERS**

Editor: Wolfgang Banzhaf

## ***GENETIC PROGRAMMING* BOOK SERIES FROM KLUWER ACADEMIC PUBLISHERS**

Editor: John Koza

[koza@stanford.edu](mailto:koza@stanford.edu)

## GP MAILING LIST

To subscribe to the Genetic Programming e-mail list,

- send e-mail message to:

[genetic\\_programming-subscribe@yahoogroups.com](mailto:genetic_programming-subscribe@yahoogroups.com)

- visit the web page

[http://groups.yahoo.com/group/genetic\\_programming/](http://groups.yahoo.com/group/genetic_programming/)

## INTERNATIONAL SOCIETY FOR GENETIC AND EVOLUTIONARY COMPUTATION (ISGEC)

For information on ISGEC, the annual GECCO conference, or the bi-annual FOGA workshop, visit

[www.isgec.org](http://www.isgec.org)

## FOR ADDITIONAL INFORMATION ON THE GP FIELD

Visit

<http://www.genetic-programming.org>

for

- links computer code in various programming languages (including C, C++, Java, Mathematica, LISP)
- partial list of people active in genetic programming
- list of known completed PhD theses on GP
- list of students known to be working on PhD theses on GP
- information for instructors of university courses on genetic algorithms and genetic programming